Republic of Iraq

Ministry of Higher Education & Scientific Research

University of Babylon

College of Education for Pure Sciences

Department of Mathematics

Development of The Mathematical Optimization Algorithms for The Neural Networks

A Thesis

Submitted to the Council of College of Education for Pure Sciences,

Babylon University in Partial Fulfillment of the Requirements for

the Degree of Master in Education / Mathematics.

By

Weam Abbas Obaid Matar

Supervised by

Assist. Dr. Ahmed Sabah Al-Jilawi

2023 A.D. **1444 A.H.**

بسم الله الرحمن الرحيم

(يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنْكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ وَاللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ).

" سورة المجادلة "، آية:11

صدق الله العظيم

Supervisor Certification

I certify that this thesis entitled "Development of The Mathematical Optimization Algorithms for The Neural Networks" was prepared by the student "Weam Abbas Obaid Matar" under my supervision at the University of Babylon, College of Education for Pure Sciences as a partial fulfillment of the requirement the Degree of Master in Education /Mathematics.

Signature:

Name: Dr. Ahmed Sabah Al-Jilawi

Scientific Grade: Assistant Professor

Date: / / 2023

In view of the available recommendation, I forward this thesis for debate by the examining committee.

Signature:

Name: Dr. Azal Jaafar Musa

Scientific Grade: Assistant Professor

Address: Head of Mathematics Department

Date: / / 2023

Scientific Supervisor's Certification

This is to certify that I have read this thesis entitled "Development of The Mathematical Optimization Algorithms for The Neural Networks" and I found it is qualified for debate.

Signature:

Name: Dr. Ahmed Abbas Mohammed

Title: Professor

Address: College of Education for Pure Sciences, University of Kufa

Date: / / 2023

Scientific Supervisor's Certification

This is to certify that I have read this thesis entitled "Development of The Mathematical Optimization Algorithms for The Neural Networks" and I found it is qualified for debate.

Signature:

Name: Dr. Hussien Ali Mohammed

Title: Assist Professor

Address: College of Education for Pure Sciences, University of Karbala

Date: / / 2023

Linguistic Reviewer's Certification

This is to certify that I have read this thesis entitled "Development of The Mathematical Optimization Algorithms for The Neural Networks" and I found it is qualified for debate.

Signature:

Name: Prof. Raad Kareem Abd-Aun

Title: Professor

Address: English Department, University of Babylon

Date: / / 2023

Examining Committee Certificate

We certify that we have read this thesis entitled "Development of The Mathematical Optimization Algorithm for The Neural Networks" as an examination committee, examined the student "Weam Abbas Obaid Matar" in its contents and that in our opinion it meets the standard of a thesis for the Degree of PhD in Education/ Mathematics.

Signature:
Name: Dr. Audia Sabrie Abdu Al-Razzaq
Title: Emeritus Professor
Date: / / 2023
Chairman

Signature:
Name: Dr. Alan Jalal Abdu Al-Kader
Title: Assistant Professor
Date: / / 2023
Member

Signature:
Name: Dr. Ali Hussein Mahmood
Title: Assistant Professor
Date: / / 2023
Member

Signature:
Name: Dr. Ahmed Sabah Al-Jilawi
Title: Assistant Professor
Date: / / 2023
Member / Advisor

Approved by the Dean of College

Signature:
Name: Dr. Bahaa Hussein Salih Rabee
Title: Professor
Address: Dean of the College of Education for Pure Sciences
Date: / / 2023

ABSTRACT

Artificial neural networks are a modern and sophisticated methodology that has attracted the interest of many researchers in various fields, including engineering, information technology, medicine, statistics and operations research, and is gaining increasing importance due to its great flexibility compared to traditional methods, as well as its ability to learn and adapt to any model.

Neural networks are usually trained using gradient-based and iteration methods to try to minimize the cost function, where the model adjusts its weight and slope, relying on the cost function and enhanced learning (supervised learning) to reach the convergence point (optimal value).

This study took two aspects, the first is the study of linear neural networks, the simplest types of which are used in the study (linear regression, multiple linear regression, polynomial regression and logistic regression) using the method of iterative gradient descent and normal equation as an analytical approach to optimization or as an alternative to gradient descent in order to reduce the cost function, reaching the minimum convergence point.

As for the other side of the study, it is how to improve and develop neural networks algorithms by making comparisons with each other and choosing the best ones in terms of speed in performance, low costs, the most accurate and the best results as we analyzed them and showed the strengths and weaknesses of each of them. Adam has been shown to train the model well because it depends on the power centers in the previous algorithms, his results were calculated using Python by applying four programs at once, where the first program shows a three-dimensional drawing of the objective function, the second program shows the shape of the compressed function from above and indicates color gradients as constraints of the function, then prints the results and displays the solution as white dots in the search path, where each point represents a solution obtained by applying the algorithm.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table 1: **Notations Used in the Thesis**

$f(x, y)$	The Objective function
$Minf(x, y)$	The Minimum of Objective Function
$Maxf(x, y)$	The Maximum of Objective Function
$f_j(x)$	The Inequality Constraint
$f_i(x)$	The Equality Constraint
MSE	Mean Squared Error
$j(\theta_0, \theta_1)$	Cost Function
LP	Linear Programming
NLP	Non-Linear Programming
IP	Integer Programming
KKT	Karush-Kuhu-Tucker Conditions
∇f	Gradiant
l	Lagrange Function
$ANNs$	Artificial Neural Networks
RMS	Root Mean Square
GD	Gradient Descent
$ReLU$	Rectified Linear Unit
NAG	Nesterov Accelerated Gradient

ACKNOWLEDGEMENTS

Thank Allah, Lord of the worlds, thank you very much and thank you always for your bounty and grace, and Allah's prayer and peace for the most honorable creation of Muhammad and Ahl al-Bayt.

I am pleased and honored to extend my sincere thanks and gratitude to my Distinguished Supervisor Dr. Ahmed Sabah Al-Jilawi, Head of the Department of Mathematics, Basic Education, who supervised this thesis, and his scientific guidance and moral support had a great impact on the production of this thesis as required. May Allah protect and according to to serve the students of science.

Also, the duty of fulfillment invites me to express my thanks and appreciation to the department of the Mathematics for providing me with the right help and advice during my scientific career.

I would also like to extend my thanks and appreciation to the chairman of the Discussion Committee and its members for graciously and accepting to discuss this letter, and express my pride in their valuable observation and comments that will culminate this effort. May Allah reward them well.

I also extend my thanks and appreciation to my colleagues in graduate studies, wishing them success and success in life . . .

DEDICATION

To from I missed him at such moments and did not give him time to learn from his experience

My Father (may Allah have mercy on him)

To the White hand that took care of me and pledged me to grow, give, enrich my soul with a flow of giving, illuminate my mind with all noble meanings and showered me with abundant knowledge, taught me that knowledge grows better in this world and the afterlife and the value of life with its goals and striving to achieve them, and Life is Love, work and giving to

My Mother

To my ideals, comrades of the road and struggle to

My Sisters and Brothers

To those with whom I know the meaning of life, to those who stood by me and supported me

My Husband and Children

Weam Abbas Obaid

PUPLICATIONS

1-Solving Numerical Optimization Model of Neural Network. Journal of Survey in Fisheries Sciences (JSFS) with E-ISSN: 7487-2368. Vol. 10, No. 3S (2023).Special Issue 3.

2-Study the Neural Network Algorithms of Mathematical Numerical Optimization. Mustansiriyah Journal of Pure and Applied Sciences, on May, 2023 (Volume 1, Number 2).

CHAPTER 1

LITERATURE REVIEW

1.1 Related Work

The previous works that are detailed in the framework of this thesis are split into eight main categories.

1. Applied Mathematics.
2. Mathematical Optimization.
3. Numerical Optimization.
4. Algorithms.
5. The Optimization Algorithm.
6. Neural Networks Optimization.
7. Constrained and Unconstrained Optimization.
8. Optimization Algorithms of Neural Networks.

1.2 Applied Mathematics

The use of mathematical techniques in a variety of disciplines, including physics, engineering, medicine, biology, finance, business, computer science, and industry, is known as applied mathematics. As a result, applied mathematics combines specific expertise with mathematical science. The phrase "Applied Mathematics" also refers to a line of work where mathematicians create and analyze mathematical models to address real-world issues. In the past, the development of mathematical theories was sparked by real-world applications these ideas then became a subject of study in pure mathematics, where abstract concepts are examined in depth. Therefore, research in pure mathematics is strongly tied to the practice of applied mathematics [42]. In the past,

differential equations and applied analysis were the main components of applied mathematics. Applied probability and Theory of approximation (broadly construed to encompass variable techniques, numerical analysis, and asymptotic methods) [34]. The development of Newtonian physics is strongly tied to these fields of mathematics, and up to the middle of the 19th century, there was little clear separation between mathematicians and physicists. This day had an impact on American education for decades since, until the early 20th century, instead of physics departments, applied mathematics departments in American universities continued to teach fluid mechanics and classical mechanics [42]. Applied Mathematics has historically been used in the engineering and computer science departments. It is common for optimization issues to have multiple viable solutions, or to be multimodal. There may be a combination of universally good and locally good solutions, or they may all be universally good (the same value of the cost function) [51].

The iterative nature, traditional optimization approaches may not perform effectively when used to generate various solutions because it is not always possible to do it, even when the algorithm is performed multiple times with different starting points.

To address optimization problems, it is frequently required to use optimization techniques that allow for the minimization or maximization of specific objective functions [50]. Non-linear or polynomial issues occasionally require approximation since they cannot be solved accurately. In these cases, heuristics that can deal with difficulties of this nature must be applied [45]. In some algorithms, the solution is approximated using evolutionary algorithms, while in others, the constraints and objective functions are linearized at a specific location in the space using partial and full derivatives.

It is suggested to employ artificial neural networks to simulate target function in optimization settings so that different problem-solving strategies can be used [3], the core of deep learning algorithms are neural networks, sometimes referred to as (ANNs) artificial neural networks or (SNNs) simulation neural networks since they replicate how actual neurons communicate with one another, their name and structure are both

borrowed from the human brain [55]. ANNs are composed of three layers of nodes: the input layer, output layer and one or more hidden layers, each node that is coupled with another node has loads and limitations connected to it. When its output reaches the predetermined limit value, the node activates and begins transferring data to the next level of the network. If not, it won't be sent [1]. Training data is essential for neural networks to develop and enhance accuracy over time. These learning algorithms can, however, be useful tools in computer science and artificial intelligence for rapidly detecting and aggregating data. Speech recognition and image recognition, for example, are quicker than manual identification [46].

1.3 Mathematical Optimization

In one of the earliest instances of optimization theory, Archimedes correctly theorized that the ideal curve is a semicircle. This example deals with finding a geometric curve of a certain length that will, when coupled with a straight line, enclose the most practicable area. Many ideas that were proposed in the early study attempt to define and characterize natural events [54].

One of the instances came roughly a century after Heron of Alexandria's theory that light always takes the quickest route. made his Archimedes Conjecture. It wasn't until fermat successfully expanded this postulate in 1657 by asserting that light always follows the path that involves the least amount of time as opposed to the least amount of distance [50]. The selection of a function that minimizes specific functions is the core issue in another field of optimization (certain category of functions the sphere of which is a set of functions of real value). Newton was aware of two problems that can be compared, while moving through the air at a constant speed in the first case, the curve should be selected so that the solid substance that results from its rotation around a line through its endpoints encounters the least amount of resistance [53].

The brachistochrone is the second issue, two points in space are offered in this puzzle. A frictionless bead traveling along the curve from one point to the other must discover the shape of the curve that will allow it to complete the journey in the shortest amount of time. In 1696, John Bernoulli offered this problem as a rival; it was subsequently solved by Bernoulli, Dei'hopital, Lebniz, and Newton (who only needed a day). In addition, at the period of Euler, several laws of mechanics were initially expressed in terms of principles of optimality (examples include the lagriange's kinetic laws, the least action principle of Maupertuis, and the concept of least restraint of Gausse) [42]. Gausse and Lagrange both contributed in other ways. Lagrange developed his lagrange multipliers in 1760 and used them to create a strategy for resolving optimization issues with equality constraints [41]. Examining a function's behavior towards its optimum is another use for the Lagrainge transformation. The least squares curve fitting method was developed by Gausse, who made contributions to numerous domains and is important to both statisticians and optimizers [55].

When he created R. Hamilton a set of functions known as Hamiltonian in 1834, it was used to formulate an optimal principle combining what was known at that time about optics and mechanics. J. W. Gibbs introduced an additional optimization principle in 1875 that was focused on the equilibrium of a thermodynamic system. Currently, there have been an increasing number of contributions. The works of the painter and Bellman were previously recognized as two of the most spectacular recent achievements. Another is the work of pontryagin and his colleagues in 1962 who created the maximal principle that is applied to solving problems in the theory of optimal control [50].

Optimization is a crucial mathematical technique that seeks to determine the best value of variables that produce the minimum or maximum values for a mathematical function, it has been one of the most important and effective tools in our daily lives [54].

In mathematics, optimization is the minimization or maximization of a function that has changeable constraints, the notation the employ is as follows:

1. The objective function f, which be to maximize or minimize is a (scalar) function of x.

2. The vector of variables, often known as the parameters or the unknowns, is termed x.

3. The constraint functions $f_i(x)$ and $f_j(x)$ establish precise equations and inequalities that the unknown vector must satisfy [50].

A mathematical optimization problem, or simply an optimization problem, consists of objective function and constraints, its general formula is as follows; $f : X \to R$

$$\begin{cases} \text{Maximize / minimize} & f(x) \\ \text{Subject to} & f_j(x) \geq 0, \quad (j = 1, \cdots, K) \\ & f_i(x) = 0, \quad (i = 1, \cdots, L) \\ & x = (x_1, \cdots, x_n)^T \in R^n \\ & x \geq 0 \end{cases} \tag{1.1}$$

$f(x)$, $f_i(x)$ and $f_j(x)$ are scalar function of the real column vector x, x_i is the components of $x = (x_1, \cdots, x_n)^T$ are called decision variables or feasible set, they can either continuous, intermittent or mixed the vector x, is called a decision vector which of $n-$ dimensional space R^n [55].

There are $K + L$ constraints in total, with $f_i(x)$ constraints in terms of L equality and $f_j(x)$ constraints in terms of K inequality, the function $f(x)$ is known as the objective function or cost function, it is possible for the goal function $f(x)$ to be linear or non-linear, it turns into a problem with linear constraints if all of the constraints $f_i(x)$ and $f_j(x)$ are linear (a linear programming problem) [55]. The discipline of linear programming falls under optimization. The issue of maximizing (minimizing) the linear target function while subject to the constraints of linear inequality and equality on the choice variables is solved mathematically through the use of linear programming, either

Java or visual basic, python is a programming language [53] [18].

When the goal function and each constraint are linear functions of x, the problem called a linear programming problem. This type of optimization problem is perhaps the most common one to be defined and solved, especially in management, financial, and economic applications [50]. In the physical sciences and engineering, non-linear programming issues in which at least part of the constraints or the aim are non-linear functions tend to naturally occur; also increasingly used in management and economic sciences, are generally non-linear functions in this instance. A vector might be mixed, discrete, or continuous in n-dimensional space [41].

Constraints become non-linear when the function is non-linear. In some unusual circumstances, linear programming can be used such as the simplex method, where f, f_i, and f_j are linear. Because of this, the problem is now linear, when some design variables contain only discrete (integer) values while others already have continuous values, the situation is known as a mixed-type problem, these problems are usually difficult to solve. For complex optimization issues [53].

1.4 Numerical Optimization

Examining the design goal to determine whether the proposed solution is practical and/or ideal is probably the most computationally time-consuming step in solving the optimization problem. Usually, it make these assessments several times, often hundreds or even millions of times [41]. The difficulty of computing increases as each evaluation process requires, the use of an efficient algorithm will enable you to use less objective evaluations in general, it is the primary method to decrease the quantity of these evaluations necessary [4]. This is typically not an option. Also employ some, approximation approaches to estimate the objectives or to build an approximate model to forecast the effects of the solution without actually employing the solution. Another

option is to replace the low-resolution model of the original objective function, derived through computer simulation using a coarse interest structure architecture [54].
The original model should be corrected because a low-resolution model is faster, but less accurate from the original model. To obtain the optimal design at a low computational cost, special techniques should be used to apply an approximate model or low-precision correction in the optimization process [18].

1.5 Algorithms

Algorithms are a limited collection of precise instructions that can be carried out by a computer and are used to solve a set of issues or carry out calculations. Algorithms are used to execute calculations, data processing, machine thinking, and other tasks [15].
The idea of the algorithm existed for Babylonian mathematicians (2500 BC.M), Egyptian mathematicians (1550 BC.M). Arab mathematicians such as Al-Kindi used the cryptographic method to decipher codes based on frequency analysis in the ninth-century. Greek mathematicians also employed the Euclidean algorithm and algorithms to determine prime numbers and the greatest common factor between two integers in the ninth century, mathematician Mohammad ebn Musa al-Khwarezmi is where word "al-Khwarizmi" originates [27]. It is usually assumed that these instructions are explicitly listed and described by the controls as starting from the top and flowing down because an algorithm is an exact list of instructions and specific steps in a certain computer order, which is extremely important for the algorithm to always perform correctly. The most common definition of algorithms is that they are commands given to the computer in the form of "if so" with the addition of some descriptions and specific conditions for the implementation of each task. Algorithms have been described in computer science as a programmatic construction of an ordered nature [46].

1.5.1 Basic Structures in Algorithms

The meaning of the algorithm is limited to only three compositions, which are the basic compositions, which are [53].

- ***Sequence:.*** Means several sequential and regular steps.
- ***Testing or making decisions:.*** Some issues can't be resolved by following a clear set of steps. Instead, you might have to put some conditions to the test, examine the results, follow a path with sequential instructions if the result is correct, and follow a another path with instructions if the result is erroneous [27].
- ***Repetition:.*** When attempting to solve an issue, it may be necessary to go through the same set of actions repeatedly until you find the desired answer. This is what is meant by repetition [27], see figure (1.1)

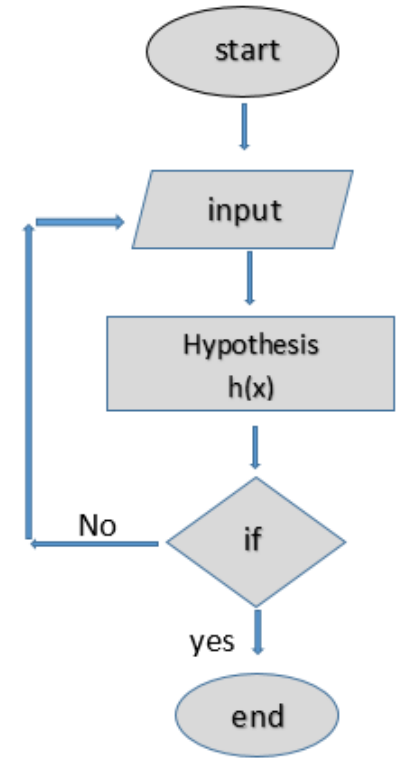


Figure 1.1: The Flowchart of the Algorithm

1.6 The Optimization Algorithm

In applied mathematics, optimization is a particularly important topic of study. The goal of optimization algorithms is to determine the parameters of a system that perform the best under different scenarios, the initial step in solving an optimization problem is to determine the objective function that specifies the connections between the system constraints and system parameters [15]. Many factors might cause the objective function to be non-linear, complicated, or non-differentiable. Real- world optimization issues often contain limitations, and the optimal value of the independent variable almost always occurs at the limit of the restriction in real world optimization problems, which is not surprising because we normally hope to receive the "best" [4]. The process of determining the best overall solution to an optimization problem is referred to as global optimization. The worldwide optimum of an objective function is often defined as the global minimum of an objective function, where as the global minimum of an objective function is frequently used to define the global optimum of an optimization problem [44]. Resilience, flexibility to many issue models, speedy determination of the global minimum value with minimal control parameters, and affordable processing should all be present in optimization algorithms [45].

1.7 Neural Networks Optimization

Neural networks are systems with interconnected nodes (Mathematical model) consisting of many layers of components that perform calculations simultaneously. The first time such a structure was developed, the (Neurons) of the smallest computational units of the human (brain) were used as a scale [57]. Neurons are another name for the smallest computational units in an artificial neural network. The input layer, the hidden layer (or layers) and the output layer are often located in neural networks [1], these layers work

together to recognize hidden patterns in the raw data, group the data into groups, and then classify the data, with network experience again, its performance improves [56].
The cost function must be specified while training a neural network model in order to quantify the discrepancy between the actual values and the desired outcomes. This will allow the network to produce an accurate prediction, which will result in a drop in the cost function's value. The goal of optimizing the training procedure is to decrease (or increase) the cost function. Optimization algorithms are essential tools in applied mathematics [46]. The weights (w) are one of the internal parameters that are used to calculate the output values and play a crucial and effective role in training the neural network model and producing accurate results. In order to update the model training parameters to their ideal values, a variety of optimization procedures are applied [52]. see figure (1.2)

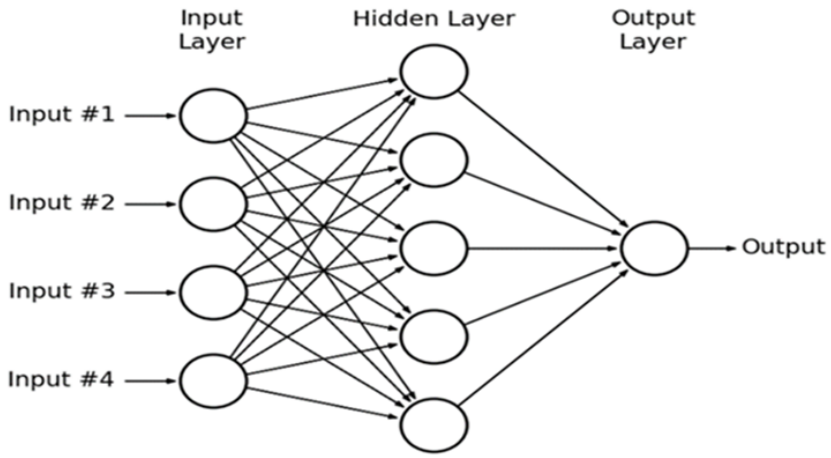


Figure 1.2: The Neural Network Model
Shows the Neural Network layers (nodes), input layers, hidden layers and output layer.

Training neural network models and getting better results heavily depend on the internal model parameters (weights and deviations), which are used to generate the output values the network parameters that affect model training and model output must be updated and calculated using a variety of optimization approaches and algorithms in order for them to converge or reach optimal values [11]. Optimization techniques can help the model provide better and quicker results by modifying the model update

strategy for weights and bias parameters, in deep learning, the idea of loss indicates how poorly the model is performing at that particular moment. As a result, this loss should be used to train the network to perform better, that is, accept the loss and try to reduce it because less loss indicates that the model will perform better [32] [22].

Similar to the optimal weights, they cannot be known at the outset but can be determined through some trial and error depending on the loss function. Optimizers utilize optimization techniques to minimize losses and produce the most accurate results possible [36].

1.8 Constrained and Unconstrained Optimization

Suppose that the optimization problem;

$$\begin{cases} \text{Min} & f(x) \\ \text{S.t} & x \in S \end{cases}$$

$f : R^n \to R$ the objective function is what we want to minimize or cost function, a real-valued function, and the vector x is an independent variable; $x = (x_1, \cdots, x_n)^T \in R^n$, the variables $(x_1, \cdots, x_n)$ are said to be decision variables, $S \subseteq R^n$ is said to be constraints or feasible set [11].

The above optimization problem can be understood as finding the "best" vector x out of all possible vectors in x. The vector that yields the target function's smallest value is the one we mean when we say it is the best vector. As there are multiple minimizers in this situation, finding any of them will suffice. Moreover, some optimization issues demand that the goal function be maximized, in this case we are looking for optimization tools [11]. Extremists are another name for mini-amplifiers. However, because maximizing (f) is equivalent to minimizing $(-f)$, the maximization problems can be equivalently written in the form of minimization [45].

On the minimization problem without losing the ability to generalize. Because the decision variables are restricted to being in the constraint set S, the problem above is a general type of constrained optimization problem. If $S = R^n$, we call the issue a constraint-free optimization problem. The restriction (x *belong to* S) is referred to as a set constraint when the issue is being discussed as an constrained optimization problem. Takes form $S = \{x : \ f_i(x) = 0, \ f_j(x) \geq 0\}$ where f_i, f_j are given function these restrictions are referred to as functional limits. If $S = R^n$ is referred to as unconstrained [37].

1.9 Optimization Algorithms of Neural Networks

Optimization algorithms are important tools in Applied Mathematics , in order to train a neural network model to produce an accurate prediction that leads to a decrease in the value of the cost function, it determine the discrepancy between the actual and expected values [48].

The parameters that affect the training of the model are updated using various optimization techniques until the model reaches optimal values, where the weights (w) are one of the internal parameters that are used to calculate the output values and play a crucial and effective role in training the neural network model and achieving accurate results, it is an optimization algorithm for neural networks [36], see figure (1.2)

In 1943, McCulloch (a scientist specializing in neuro-science) and Bates (in mathematics) created a neuron model that acts as a switch in an electrical circuit that receives inputs from other neurons and the activity of the cell depends on the total weight of these inputs [1].

This neural network's initial objective was to develop a computer system that could mimic the human brain while solving problems, so neural networks important for their remarkable ability to extract meaning from complex and inaccurate data, which gives them the ability to understand patterns and observe tendencies that neither humans nor

other computing technologies can notice, so it help humans solving complex problems in their daily lives [49]. These networks can also learn and model complex and non-linear relationships between data inputs and outputs, in addition, it make generalizations and inferences to reveal hidden relationships between inputs and outputs, patterns and predictions [52].

Now a system that learns to make predictions by doing these things is called a neural network.

- Entering data.
- Setting expectations.
- Comparing the predicted results to the desired results.
- Changing the vectors to make future predictions more accurate.

Knowing when to end training and what the accuracy target should be established is a crucial aspect of training neural networks. The procedure continues until the discrepancy between the expectation and the required goals becomes a little [56].

CHAPTER 2

BASIC CONCEPT

2.1 Introduction

One of the simplest and most effective instruments in our daily life has been optimization. Finding the best value for variables that yield the lowest value or maximum value of a mathematical function (objective function) is the goal of optimization [55]. In mathematical programming optimization algorithms are a vital and useful technique for finding a solution. Given the first estimate of the value of the variables, the computer will usually use optimization algorithms to create a series of optimized estimates, or iterations, until the optimal solution is reached (or generates a good approximation to the optimal value), the general formula of optimization problem is as follows [51] [23];

$$\begin{cases} Maximize(minimize): & objective\ function \\ subject\ to: & constraints \end{cases} \tag{2.1}$$

2.2 Linear Programming (LP)

The collection of optimization models known as linear programming (LP) models, which have polyhedral feasible zones and linear objective functions, are quite helpful in practice. This is because linear programming models, which only have two variables, can accurately capture many real-world issues (even if multiple approximations must be made first). Additionally, and perhaps more crucially, there are efficient algorithms available for solving linear programming.

Consequently, the issue can be resolved graphically [20], the optimization problem in general take form;

$$X \in R^n, f, f_i : R^n \to R,\ R \cup \{\pm\infty\}$$

$$\begin{cases} \text{Maximize (minimize):} & f(x) & \textit{objective function} \\ \text{Subject to:} & f_i(x) = k_j,\ j \in K & \textit{equality constraints} \\ & f_i(x) \geq c_i,\ i \in L & \textit{inequality constraints} \\ & x \in R^n,\ x \geq 0 & \end{cases} \tag{2.2}$$

Now: objective function of linear programming (LP):

$$f(x) = a^T x = \sum_{i=1}^{L} a_i x_i, (a \in R^n)$$

constraint function: $f_i(x) = a_i^T x - k_j\ (a_i \in R^n,\ k_j \in R,\ i, j \in L \cup K)$

$$X = \{x \in R^n |\ x_i \geq 0,\ i = 1, \cdots, L\}$$

2.3 Non-linear Programming (NLP)

Some functions $f_i, h_i\ (i \in L \cup K)$ are non-linear, there are two types of non-linear programming problems [13];

1. Single variable:

   1- unconstraint: $min(max)\ w = f_i(x)\ subject\ to\ (-\infty \leq x \leq \infty)$

   2- constraint: $min(max)\ w = f_i(x)\ subject\ to\ (c \leq x \leq d)$

2. Multivariable:

   1- Multivariable with constraind: $min(max)\ w = f_i(x_1, \cdots, x_n),\ x_i \geq 0$

2- Multivariable with equality constraints:

$$\begin{cases} \min\ (\max) & w = f_i(x_1, \cdots, x_n) \\ \text{subject to} & h(x_i) = 0 \\ & x_i \geq 0 \end{cases}$$

3- Multivariable with inequality constraints:

$$\begin{cases} \min\ (\max) & w = f_i(x_1, \cdots, x_n) \\ \text{subject to} & h(x_i) \geq 0, h(x_i) \leq 0 \\ & x_i \geq 0 \end{cases}$$

4- Multivariable with constraints equality and inequality [23]:

$$\begin{cases} \min\ (\max) & w = f_i(x_i) \\ \text{subject to} & h(x_i) = 0 \\ & h(x_j)(\leq, \geq)0,\ i, j = 1, \cdots, n \\ & x_i, x_j \geq 0 \end{cases}$$

Now; Unconstrained Optimization: $L \cup\ K = \phi,\ X = R^n$

Constrained Optimization: $L \cup\ K \neq \phi,\ and/or\ X \subseteq\ R^n$

2.4 Integer Programming (IP)

$X \subseteq \{0, 1\}^n$ (binary) or $x \subseteq Z^n$ (integer)

The constraint shows that integer programming is a certain case of non-linear programming $x_j \in \{0, 1\}$ can be written as [2]; $x_j(1 - x_j) = 0$, if a non-negative integer variable x_j is upper bounded by the integer N, it is also possible to write

$\prod_{k=0}^{N}(x_j - k) = (x_j - 0)(x_j - 1)\cdots(x_j - N) = 0$, by which we restrict a continuous variable x_j to be integer-valued [2]. The non-linear constraints, linear functions are a special kind of non-linear functions, making it clear that linear programming is a particular instance of non-linear programming. The following approach might be used to depict their relationship [7], see figure(2.1)

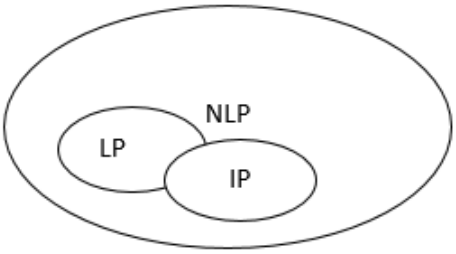


Figure 2.1: Relation between LP, IP and NLP
Non-linear programming includes linear programming and integer programming

2.5 Mathematical Optimization Classifications

The classification of optimization problems is one of the key steps to getting the desired outcome with the least amount of effort, which can be performed based on the goals, the number of constraints, available space, objective functionality, design variables and computer effort [7]. When classifying optimization problems in terms of the number of goals, two types: they have one goal, $B = 1$ and multiple goals $B > 1$ but in terms of constraints some problems do not have constraints, if $l = k = 0$ they are referred to as unrestricted problems, if $k \geq 1, l = 0$ is said to be the equality constraint problems, while $l \geq 1, k = 0$ is said to be the inequality constraint problems [55]. Also classed as linear or non-linear is the objective function; however, if both constrained and objective function are linear, the issue is categorized as a linear programming issue, although the term programming has nothing to do with computer programming [55], although the

majority of situations in real life are nonlinear, therefore in general, if constrained and objective function are non-linear, the optimization problem is non-linear optimization, on the other hand, is classified according to shape of landspace of objective function; for one goal function, the shape or landspace may differ significantly for the non-linear case if there is one valley or one peak with one global optimum, the problem is called unimodal. Which implies that the local maximum is also the global maximum. For instance, $f(x, y) = x^2 + y^2$ is a function have a local minimizer at $(0, 0)$ which is also the global minimizerl [55]. In terms of variables, they are classified into discontinuous, continuous and mixed variables, in terms of determinacy classified into deterministic for each of the above and stochastic [4], following diagram shows this classification;

$$
\textit{optimization problems}\begin{cases}
\textit{objective}\begin{cases}\textit{single objective}\\ \textit{multiobjective}\end{cases}\\
\textit{constraint}\begin{cases}\textit{constraint}\\ \textit{unconstraint}\end{cases}\\
\textit{function}\begin{cases}\textit{non}-\textit{linear}\begin{cases}\textit{quadratic}\\ \vdots\end{cases}\\ \textit{linear}\end{cases}\\
\textit{landspace}\begin{cases}\textit{unimodal(convex)}\\ \textit{mulimodal}\end{cases}\\
\textit{determinacy}\begin{cases}\textit{determinstic all the above}\\ \textit{stochastic}\end{cases}\\
\textit{variables}\begin{cases}\textit{continuous}\\ \textit{mixed}\\ \textit{discrete}\begin{cases}\textit{integer}\\ \ldots\end{cases}\end{cases}
\end{cases}
$$

Definition 2.5.1. ***Feasible solution*** [13]

A ***feasible solution*** is one that satisfies all constraints or a practical option that yields the best possible value of the objective function.

Definition 2.5.2. ***Feasible set or Feasible region*** [13]

The ***feasible set*** or ***feasible region*** is the set of all feasible solutions or is the set of all points whose coordinates satisfy the constraints of a problem.

Definition 2.5.3. ***Optimal solution*** [16]

Optimal solution is the feasible solution that we get the best objective function value possible.

Example 2.5.1. Consider the following simple linear programming problem:

$$(LP)\begin{cases} \text{maximize} & 3x+5y \\ \text{subject to} & x+y \leq 4 \\ & x+3y \leq 6 \\ & x,y \geq 0 \end{cases}$$

$(0,0) \Rightarrow 3(0)+5(0)=0$

$(0,2) \Rightarrow 3(0)+5(2)=10$

$(4,0) \Rightarrow 3(4)+5(0)=12$

$(3,1) \Rightarrow 3(3)+5(1)=14$

Therefore the maximize objective value is 14 at the point $(3,1)$, see figure(2.2)

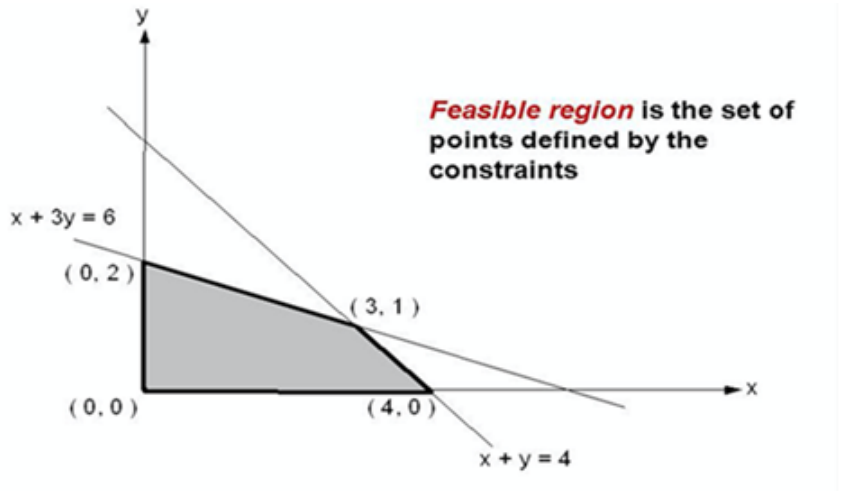


Figure 2.2: The Feasible Region of two Constraints
The figure represents the feasible region, constraints, feasible solutions of the objective function and the optimal solution.

2.6 Convex Sets and Convex Functions

Definition 2.6.1. ***Convex Set*** [41]

Suppose that S is a set $S \subseteq R^n$, S is ***Convex Set*** if the following condition is hold

$$\delta x + (1-\delta)y \in S,\ 0 \leq \delta \leq 1, \forall x, y \in S$$

i.e, if any two points x *and* y lie on a line segment lies in S. The set S is ***Non-convex*** if this condition is false, see figure(2.3)

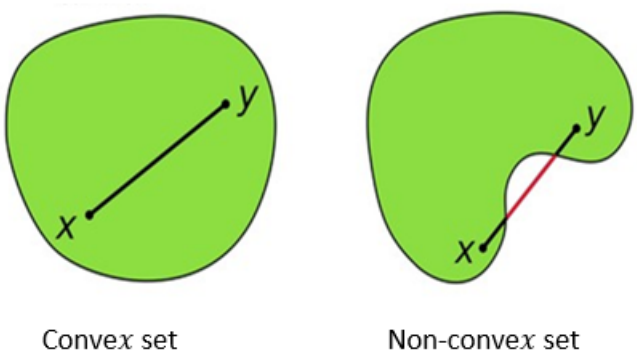


Figure 2.3: Convex and Non-convex set

Definition 2.6.2. ***Convex Function*** [2]

A set B is a nonempty convex set, where $B \subseteq R^n$, $g : B \to R$, g is said a ***Convex Function*** on B if satisfy;

$$g(\sigma x_1 + (1-\sigma)x_2) \leq \sigma g(x_1) + (1-\sigma)g(x_2), \forall x_1, x_2 \in B, \forall \sigma \in [0,1]$$

If the aforementioned inequality holds for all $x_1 \neq x_2, \forall \sigma \in (0,1)$ as a rigorous inequality, then g is referred to as a strictly convex function on S, $if\ \exists\ c > 0, \ni \forall x_1, x_2 \in B$, *and* $\forall \sigma \in [0,1]$, c is constant.

$$g(\sigma x_1 + (1-\sigma)x_2) \leq \sigma g(x_1) + (1-\sigma)g(x_2) - \frac{1}{2}c\sigma(1-\sigma) \parallel x_1 - x_2 \parallel^2$$

Then g is said to be ***Strongly convex function*** on S.

Now if (g) is ***Convex*** then $(-g)$ is ***Concave*** [2], see figure(2.4)

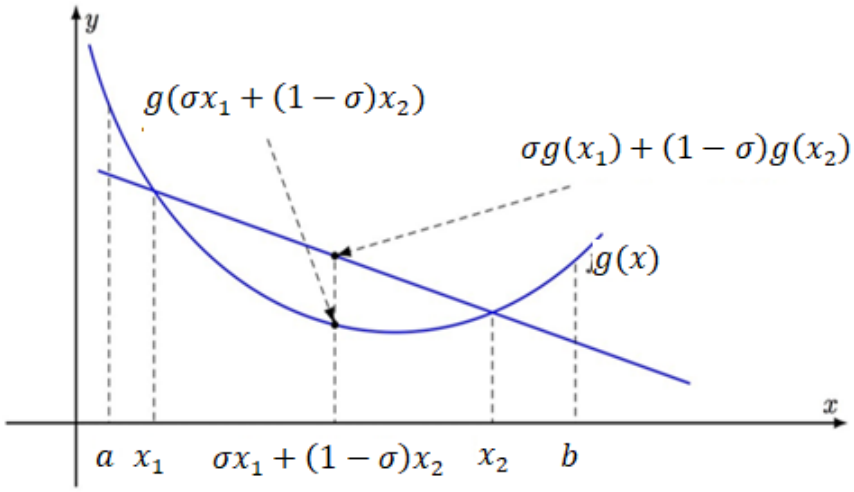


Figure 2.4: Convex Function
Convex Function if the hypotenuse connecting two points on its graph is never less than the graph.

2.7 The Vector and Matrix

2.7.1 Vectors [58]

A vector x is an ordered collection of scalars $x_i,\ i = 1, \cdots, n$. Here, n- dimension of the vector x, x_i it is known ith element of the vector x, where x is represented by a column vector, $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$. Often, it is useful to write x as a row vector, $x^T = [x_1, x_2, \cdots, x_n]$, where x^Tdenotes the transpose of x. In other word, the transpose of given column vector

x is a row vector with corresponding elements [20].

2.7.2 Two Vectors [2]

Suppose that $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$. If the corresponding elements of two vectors are equal, then the vectors are equal, that is $x_i = y_i, \forall i = 1, \cdots, n$ [2].

2.7.3 Vector Addition [41]

Suppose that ν, μ, ω vectors n-dimensional. The sum of ν and μ is denoted by $\nu + \mu$, is the vector. $\nu + \mu = [\nu_1 + \mu_1, \cdots, \nu_n + \mu_n]^T$

The following characteristics apply to addition of vectors:

1- process $\nu + \mu = \nu + \mu$ ***(Commutative)***

2- process $(\nu + \mu) + \omega = \nu + (\mu + \omega)$ ***(Associative)***

3- A zero vector $0 = [0, \cdots, 0]^T$, such that $0 + \mu = \mu + 0 = \mu$.

A vector $[\nu_1 - \mu_1, \cdots, \nu_n - \mu_n]^T$, is called difference between ν, μ denoted by $(\nu - \mu)$ the vector $(0 - \mu)$ is denoted by $(-\mu)$ that is [20];

$$(\nu - \mu) + \mu = \nu$$

$$\nu = -(-\nu)$$

$$\nu - \mu = -(\mu - \nu)$$

Vector Equation of Line: $\overrightarrow{\omega} = \overrightarrow{\nu} + \lambda \overrightarrow{\mu}$ or $\overrightarrow{\omega} = \overrightarrow{\nu} + \lambda(\overrightarrow{\mu} - \overrightarrow{\nu})$

2.7.4 The Vector Multiplication by Scalar [20]

A vector $x \in R^n$ multiplication by a real scalar $\beta \in R$ as, $\beta x = [\beta x_1, \cdots, \beta x_n]^T$, this process has following properties:

1-Process is distributive: for any scalars $\sigma, \beta \in R$

$$\sigma(x + y) = \sigma x + \sigma y$$

$$(\sigma + \beta)x = \sigma x + \beta x$$

2-Process is associative: $\sigma(\beta x) = (\sigma\beta)x$

3- Scalar 1 satisfies: $1.x = x$

4- Any scalar λ is satisfies: $\lambda.0 = 0$

5- The scalar 0 satisfies: $0.x = 0$

6- The scalar(-1) satisfies: $(-1).x = -x$

Now: $\sigma.x = 0$ *if and only if* $\sigma = 0$ *or* $x = 0$

Note that: $\sigma x = 0$ is equivalent to $\sigma x_1 = \sigma x_2 = \cdots = \sigma x_n = 0$

Definition 2.7.1. ***Linearly Combination*** [11]

A set $(x_1, x_2, \cdots, x_k)$ be a vectors and $(\alpha_1, \alpha_2, \cdots, \alpha_k)$ be a set of scalars. A ***linearly combination*** of $(x_1, x_2, \cdots, x_k)$ is a vector $\mathbf{x}$ such that, $\mathbf{x} = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_k x_k$.

Definition 2.7.2. ***Linearly Dependent*** [11]

Let $(x_1, x_2, \cdots, x_k)$ are a set of vectors, it are said to be ***linearly dependent*** if and only if there exist k scalars $(\alpha_1, \cdots, \alpha_k)$ such that $\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_k x_k = 0$, and at least one of the k scalars $(\alpha_1, \cdots, \alpha_k)$ is different from zero.

Definition 2.7.3. ***Linearly Independent*** [11]

Let $(x_1, x_2, \cdots, x_k)$ are a set of vectors, $(x_1, x_2, \cdots, x_k)$ are said to be ***linearly independent*** if and only if they are not linearly dependent.

Definition 2.7.4. ***Vector Scalar Product (Dot Product)*** [34]

Suppose that ν and ω be two n-dimensional vectors. The scalar product of ν and ω is denoted by $\nu^T\omega$ or $\nu.\omega$ where, $\nu^T\omega = \nu.\omega = (\nu_1\omega_1 + \nu_2\omega_2 + \cdots + \nu_n\omega_n)$, if $\nu^T\omega = 0$, the vectors ν and ω are called ***orthogonal***.

2.8 Vector Norm [13]

Describe the Euclidean space, limited in dimensions vector space having an inner product $\langle ., . \rangle$ ***Euclidean Norm*** defined by $\| \mu \| = \sqrt{\langle \mu, \mu \rangle} = \sqrt{\mu^T.\mu}$ for all μ in vector space.

2.8.1 Inner Product [13]

The ***Inner product*** of any two real n-vectors $\mu, \nu \in R^n$ is define by

$$\langle \mu, \nu \rangle = \mu^T\nu = \sum_{i=1}^{n} \mu_i\nu_i = \mu_1\nu_1 + \mu_2\nu_2 + \cdots + \mu_n\nu_n$$

The function $\langle ., . \rangle : R^n \to R$ is an ***Inner product*** if;

1-$\langle \mu, \mu \rangle = 0, \langle \mu, \mu \rangle > 0 \Leftrightarrow \mu = 0$ ***(Positivity)***

2-$\langle \mu, \nu \rangle = \langle \nu, \mu \rangle$ ***(Symmetric)***

3-$\langle \mu + \nu, \omega \rangle = \langle \mu, \omega \rangle + \langle \nu, \omega \rangle$ ***(Additivity)***

4-$\langle \delta\mu, \nu \rangle = \delta\langle \mu, \nu \rangle, \forall \delta \in R$ ***(Homogeneity)***

2.8.2 Norm [16]

The function $\| . \|: R^n \to R$ is known a ***Norm*** if it hold the properties:

1- $\| z \| \geq 0, \forall\ x \in R^n, \| z \| = 0\ if\ and\ only\ if\ z = 0.$

2- $\| \alpha z \| = | \alpha | \| z \|, \forall\ z \in R^n$

3- $\| u + z \| \leq \| u \| + \| z \|, \forall\ u, z \in R^n$.

Vector norms come in two types l_2 and l_∞.

2.8.3 Euclidean Norm l_2

Euclidean norm of a vector x is denoted by $\| x \|$ *or* l_2 where

$$\| x \| = \| x \|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\ .$$

Euclidean norm is a measurement of the vector's "length". Take note that from the dot product's definition, the norm can also be written as [11].

$$\| x \|^2 = (x_1^2 + x_2^2 + \cdots + x_n^2) = x^T x.$$

2.8.4 The Norm l_∞

The vector's greatest component's absolute value is represented by norm in the vector x [11]. It is denoted by l_∞, where $\| x \|_\infty = max_{1 \leq i \leq n} \mid x_i \mid$.

Example 2.8.1. Let the vector A = [-3 6 2] compute the Euclidean norm of the vector.

Solution: $\| A \| = \sqrt{(-3)^2 + (6)^2 + (2)^2} = \sqrt{9 + 36 + 4} = 7$

Example 2.8.2. Let the vector $A = [-3\ 1\ 2\ -1]$, find the Norms of the vector.

Solution: $\| A \|_2 = \sqrt{(-3)^2 + (1)^2 + (2)^2 + (-1)^2} = \sqrt{15}$

$\| A \|_\infty = max(\mid -3 \mid, \mid 1 \mid, \mid 2 \mid, \mid -1 \mid) = 3$

Example 2.8.3. Let two vectors $A = \begin{bmatrix} 2 \\ -4 \\ 3 \end{bmatrix}$ *and* $B = \begin{bmatrix} -1 \\ 5 \\ 2 \end{bmatrix}$, find the Inner product.

Solution: $\langle A, B\rangle = A^T B = [2 \ -4 \ 3] \begin{bmatrix} -1 \\ 5 \\ 2 \end{bmatrix} = (2)(-1) + (-4)(5) + (3)(2) = -16$

Theorem 2.8.1. ***Cauchy-Schwarz Inequality*** *[11]*

Let $\nu, \mu \in R^n$. Inequality Cauchy-Schwarz

$$| \langle \nu, \mu \rangle | \leq \| \nu \| \| \mu \|$$

Holds. Additionally, equality exists if and only if $\nu = c\mu$ for some $c \in R$.

Proof:

Suppose that ν, μ are unit vectors; that is $\| \nu \| = \| \mu \| = 1$, then,

$$\begin{aligned} 0 \leq \| \nu - \mu \|^2 &= \langle \nu - \mu, \nu - \mu \rangle \\ &= \| \nu \|^2 - 2\langle \nu, \mu \rangle + \|\mu\|^2 \\ &= 2 - 2\langle \nu, \mu \rangle \\ \text{or} \qquad \langle \nu, \mu \rangle &\leq 1 \end{aligned}$$

equality satisfy if and only if $\nu = \mu$.

Now: suppose that nither ν nor μ is zero (since it is evident that the inequality exists if one of them is zero), swap ν and μ by a unit vectors $\frac{\nu}{\| \nu \|}, \frac{\mu}{\| \mu \|}$

Apply property(4) is obtained, $\langle \nu, \mu \rangle \leq \| \nu \| \| \mu \|$

Swap ν by $-\nu$ and again apply property(4) is obtained, $-\langle \nu, \mu \rangle \leq \| \nu \| \| \mu \|$

absolute value inequality is implied by the last two inequalities.

Equality satisfies if and only if $\frac{\nu}{\| \nu \|} = \pm \frac{\mu}{\| \mu \|}$, that is $\nu = c\mu$ for some $c \in R$. ■

2.9 The Symmetric Matrix [41]

The most typical matrices seen in numerical optimization are symmetric.

2.9.1 The Symmetric

Assume $X = [x_{ij}]$ be a matrix square, a matrix X known as ***Symmetric*** if $X = X^T$ that is $x_{ij} = x_{ji}$.

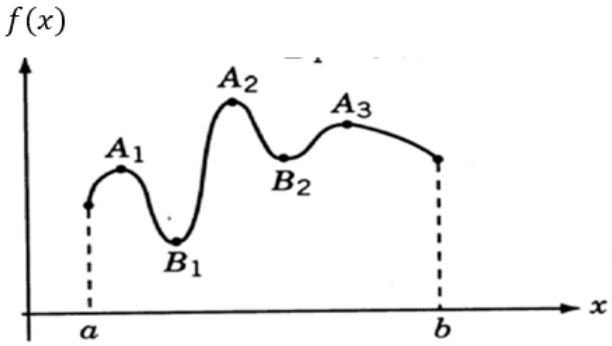
2.9.2 The Semidefinite Programming

Let A^k be a collection for all symmetric $k \times k$ matrices $A^k = \{X \in R^{k\times k} | X = X^T\}$.
The notation $X \succ 0$ $(X \succeq 0)$ it is know X is ***Symmetric and positive definite (Semidefinite)***; that is, for all nonzero $\nu \in R^n (\nu^T X \nu = \sum_{ij} X_{ij}\nu_i\nu_j \geq 0, \forall \nu \in R^n)$.
Furthermore, let A_+^m denote the set of ***Symmetric positive semidefinite*** matrices [41];

$$A_+^m = \{X \in A^m | X \succeq 0\}$$

Definition 2.9.1. ***The Local Minimum*** [50]
Assume that $x^\star \in A$ it is know ***Local minimize*** of f over A if $\exists\ \varphi > 0$
$B_\varphi(x^\star) = \{\|x - x^\star\| < \varphi | x \in R^n\}$, such that $f(x^\star) \leq f(x), \forall x \in A \cap B_\varphi(x^\star)$, a point $x^\star$ it is know ***Strict local minimize*** of f over A if $x^\star \neq x \ni f(x^\star) < f(x),\ x \in A \cap B_\varphi(x^\star)$.

Definition 2.9.2. ***The Global Minimum*** [2]
Assume that $x^\star \in A$, $x^\star$ it is know a ***Global minimize*** of f over A if satisfy
$f(x^\star) \leq f(x), \forall x \in A,\ x^\star$ it is know ***Strict global minimize*** if $f(x^\star) < f(x), \forall x \in R^n$ *with* $x^\star \neq x$, see figuer(2.5)



$B_1 = Global\ minimum.$ $A_2 = Global\ maximum.$

$B_1, B_2. = Local\ minimum.$ $A_1, A_2, A_3 = Local\ maximum$

Figure 2.5: Global and Local points

Definition 2.9.3. ***Continuous Function*** [37]

defined $h(x)$ a real valued function on a subset $S \subseteq R^n$ is called ***Continuous*** at x if given $\delta > 0$, there is $\sigma > 0$ such that $\| y - x \| < \sigma$, means $|h(y) - h(x)| < \delta, \forall y \in S$.

If $h(x)$ is continuous for every $x \in S$, then $h(x)$ is said to be continuous on S.

Definition 2.9.4. ***Gradient*** [58]

The n-dimensional vector of a first order partial derivatives of h is the ***Gradient*** of $h(x_i)$, and denpted by $\nabla h(x_i)$.

$$\nabla h(x_i) = \begin{bmatrix} \dfrac{\partial h(x_1)}{\partial x_1} \\ \dfrac{h(x_2)}{\partial x_2} \\ \vdots \\ \dfrac{\partial h(x_n)}{\partial x_n} \end{bmatrix}$$

Where $S \subseteq R^n, \forall x \in S$ *and* $\forall i = 1, \cdots, n$, $\dfrac{\partial h(x_i)}{\partial x_i}$ exists and continuous, then $h(x_i)$ is said ***Differential*** in S.

Definition 2.9.5. ***Hessian Matrix*** [41]

The ***Hessian Matrix*** of $h(x)$ is denoted by $H(x)$, is the second partial derivatives of h, it is defined by.

$$H(x) = \nabla^2 h(x) = \begin{bmatrix} \frac{\partial^2 h(x)}{\partial^2 x_1} & \frac{\partial^2 h(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 h(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 h(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 h(x)}{\partial^2 x_2} & \cdots & \frac{\partial^2 h(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 h(x)}{\partial x_n \partial x_1} & \frac{\partial^2 h(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 h(x)}{\partial^2 x_n} \end{bmatrix}$$

Definition 2.9.6. ***Descent Direction*** [37]

Assume that $h : R^n \to R$ be differentiable at $x \in R^n$, $if\ \exists\ v \in R^n$ (v is vector) such that $\langle \nabla h(x), v \rangle < 0$, then v it is know a ***Descent direction*** of h at x.

Definition 2.9.7. ***Stationary point*** [37]

Suppose that $h : R^n \to R$ be differentiable. $x^\star \in R^n$ is called ***Stationary point*** of h if $\nabla h(x^\star) = 0$.

2.10 The Optimality Conditions

The following optimality conditions determine many methods for solving systems of non-linear equations; think of an optimization issue. If $X = R^n$, i.e., minimze h without limitations, it can be said [57]; $minimize_{x \in R^n} h(x)$.

1. If h is continuously differentable, then a condition necessary for $x^\star \in R^n$ to be a problem-solving factor is $\nabla h(x^\star) = 0$.

2. If h is twice continuous differential, then a conditions necessary for $x^\star \in R^n$ to be a best value of the problem is $\nabla h(x^\star) = 0, \nabla^2 h(x^\star) \geq 0$.

3. sufficient conditions for $x^\star \in R^n$ to be a local solution of problem are

$\nabla h(x^\star) = 0, \nabla^2 h(x^\star) > 0.$

Theorem 2.10.1. ***First-Order Necessary Condition*** *[37]*

Suppose that $f_i : R^n \to R$ be differentiable. If $x^\star$ is a local minimizer of f_i, then

$\nabla f_i(x^\star) = 0.$

Proof:.

Define $g : R \to R$ as $g(\theta) = f_i(x^\star + \theta v)$ for some $v \in R^n$

Then $g'(\theta) = v^T \nabla f_i(x^\star + \theta v)$. If $\theta = 0$, we get $g'(0) = v^T \nabla f_i(x^\star)$

By definition

$$g'(0) = \lim_{\theta \to 0} \frac{f_i\left(x^\star + \theta v\right) - f_i\left(x^\star\right)}{\theta}$$

Since $x^\star$ is a local minimizer, $\exists\ \kappa > 0$, such that $f_i(x^\star + \theta v) \geq f_i(x^\star), \forall\ \kappa \geq \theta > 0$

Therefore obtained $v^T \nabla f_i(x^\star) \geq 0$

Since (v) is an arbitrary, we can replace (v) by $(-v)$

And thus $-v^T \nabla f_i(x^\star) \geq 0$

Therefore, $v^T \nabla f_i(x^\star) = 0, \forall v \in R^n$, thus $\nabla f_i(x^\star) = 0.$ ■

Theorem 2.10.2. ***Second-Order Necessary Condition*** *[37]*

Suppose that $f_i : R^n \to R$ be twice differentiable. If $x^\star$ is a local minimizer of f_i, then

$\nabla f_i(x^\star) = 0$ & $\nabla^2 f_i(x^\star)$ is positive semidefinite.

Proof:.

Let $b \in R^n$. To proof $b^T \nabla^2 f_i(x^\star) b \geq 0$. By using Taylor expansion of f_i at $x^\star$, we get

$$f_i(x^\star + \theta b) = f_i(x^\star) + \theta b^T \nabla f_i(x^\star) + \frac{\theta^2}{2} b^T \nabla^2 f_i(x^\star) b + u(\theta^2)$$

Since $\nabla f_i(x^\star) = 0$ by First-Order Necessary Condition theorem, we have;

$$f_i(x^\star + \theta b) = f_i(x^\star) + \frac{\theta^2}{2} b^T \nabla^2 f_i(x^\star) b + u(\theta^2)$$

Dividing both sides by θ^2, we have;

$$\frac{f_i\left(x^\star+\theta b\right)-f_i\left(x^\star\right)}{\theta^2}=\frac{1}{2}b^T\nabla^2 f_i(x^\star)b+\frac{u(\theta^2)}{\theta^2}$$

both sides take the limit, using the fact that $x^\star$ is a local minimizer,we get

$$0\leq\lim_{\theta\to 0}\frac{f_i\left(x^\star+\theta b\right)-f_i\left(x^\star\right)}{\theta^2}=\lim_{\theta\to 0}\left(\frac{1}{2}b^T\nabla^2 f_i\left(x^\star\right)b+\frac{u\left(\theta^2\right)}{\theta^2}\right)$$

since $\lim_{\theta\to 0}\frac{u\left(\theta^2\right)}{\theta^2}=0$ *, we get* $b^T\nabla^2 f_i(x^\star)b\geq 0$

Therefore, $\nabla^2 f_i(x^\star)$ is positive semidefinite. ■

Theorem 2.10.3. ***Second-Order Sufficient Condition*** *[37]*

Suppose that $f_i : R^n \to R$ be twice continuously differentiable. If $\nabla f_i(x^\star)=0$ & $\nabla^2 f_i(x^\star)$ is positive definite, then $x^\star$ is a strict local minimizer.

Proof:.

Since $\nabla^2 f_i(x^\star)$is positive definite and $\nabla^2 f_i$ is continuous,

we have $d^T\nabla^2 f_i(x)d>0, \forall x\in B(x^\star,\kappa)$ for some $\kappa>0$, and for any $d\neq 0$.

Now, let $d\neq 0$, such that $x^\star+dB(x^\star,\kappa)$, since $\nabla f_i(x^\star)=0$, and by using the Taylor expansion, there exists a point q that lies between $(x^\star)$ and $(x^\star+d)$ such that

$$\begin{aligned}f_i(x^\star+d)&=f_i(x^\star)+\nabla f_i(x^\star)^T+\frac{1}{2}d^T\nabla^2 f_i(q)d\\&=f_i(x^\star)+\frac{1}{2}d^T\nabla^2 f_i(q)d\\&>f_i(x^\star)\end{aligned}$$

Hence $f_i(x^\star+d)>f_i(x^\star)$, so $(x^\star)$ is a strict local minimum. ■

2.11 Duality Theorem

The duality theorem implies a relationship between the primal and dual that is known as complementary slakness. The number of variables in the dual is equal to the number of constraints in the primal and the number of constraints in the dual is equal to the number of variables in the primal. This correspondence suggests that variables in one problem are complementary to constraints in the other, for an inequality constraint, the constraint has slack. The term complementary slackness refers to a relationship between the slakness in a primal constraint and the slackness of the associated dual variable [2].

2.11.1 Relationship of Dual Minimization and Primal Maximization of Linear Problem

1-The coefficient matrices of the x_i and y_i are transposes of each other.
2-The coefficients in the objective function for the primal become the constants of the inequalities for the dual.
3-The constants of the inequalities of the primal become coefficients in the objective function for the dual [37].

Definition 2.11.1. ***Complementary Slackness*** [2]
The relation between the (P) and (D). Usually, primal and dual pair complementary slackness provided by

$$(P)\begin{cases}\text{maximize} & s^T x \\ \text{subject to} & Bx \le c \\ & x \ge 0\end{cases} \Leftrightarrow (D)\begin{cases}\text{minimize} & c^T y \\ \text{subject to} & B^T y \ge s \\ & y \ge 0\end{cases}$$

Theorem 2.11.1. ***Complementary Slackness Theorem*** *[2]*
suppose that a feasible solution to primal is x_1, a feasible solution to dual is x_2, then the

optimal of primal is x_1 *and the optimal of dual is* x_2 *if and only if* $x_i(z_i - B_i^T x_j) = 0$, $i, j = 1, \cdots, n$, *where* B_i *is the ith column of* B.

Proof:

if x_1, x_2 *are feasible we obtain* $z^T x_1 \geq (B^T x_2)^T x_1 = (x_2^T B) x_1 = c^T x_2$

x_1, x_2 *are optimal* $\leftrightarrow z^T x_1 = c^T x_2$, *in fact* $z^T x_1 = (B^T x_2)^T x_1 \Leftrightarrow x_1^T (z - B^T x_2) = 0$

Now; $x_1 \geq 0, \quad B^T x_2 \leq z, \quad x_1^T (z - B^T x_2) = 0$, *corresponds to the fact that the sum of all terms is 0, i.e.* $x_i(z_i - B_i^T x_j) = 0$ *is satisfy.* ■

Example 2.11.1.

$$(P) \begin{cases} \text{maximize} & x_1 + 2x_2 + 5x_3 \\ \text{subject to} & x_1 + x_2 + x_3 = 4 \\ & x \geq 0 \end{cases} \tag{P}$$

Solution:. The dual of problem (P) is given by

$$(D) \begin{cases} \text{minimize} & 4y_1 \\ \text{subject to} & y_1 \geq 1 \\ & y_2 \geq 2 \\ & y_3 \geq 5 \end{cases} \tag{D}$$

The solution of problem(P) is

$x_2 = x_3 = 0, x_1 = 4 \Rightarrow x_1 + 2x_2 + 5x_3 = 4$

$x_1 = x_3 = 0, x_2 = 4 \Rightarrow x_1 + 2x_2 + 5x_3 = 8$

$x_1 = x_2 = 0, x_3 = 4 \Rightarrow x_1 + 2x_2 + 5x_3 = 20$

The optimal solution of problem (P) is $x_1 = x_2 = 0, x_3 = 4$, and the optimal value is $x_1 + 2x_2 + 5x_3 = 20$; the optimal solution of the problem (D) is $y_1 = 5$, and the oplimal value is $4y_1 = 20$.

2.12 Equality and Inequality Augmented Lagrange for Linear Programming

Inequality & equality constraints, augmented lagrangian approach can be employed, and the standard linear programming problem (LPP) is given by [20].

$$(LP)\begin{cases}\text{Minimize} & \langle s, x\rangle \\ \text{subject to} & \langle b_i, x\rangle = q_i,\ i = 1, \cdots, n \\ & x_i \in R^n, x \geq 0\end{cases}$$

Where the vector $b_i \in R^n$, the numbers $q_i \in R$ and the vector $s \in R^n$. The notation $x \geq 0$ means that $x_i \geq 0, \forall i = 1, \cdots, n$.

Let R^n_+ be the set of vectors $x \in R^n$, such that $x_i \geq 0$. The Lagrange given by;

$$l_i(x_1, x_2) = \langle s, x_1\rangle + \langle x_2, q_i - \langle b_i, x_i\rangle\rangle$$

The common form of **augmented Lagrange** is

$$l_\alpha(x_1, x_2) = f_i(x_1) + \langle x_2, q_i - \langle b_i, x_1\rangle\rangle + \frac{1}{2\alpha} \parallel q_i - \langle b_i, x_1\rangle \parallel^2$$

The quadratic penalty term makes the new objective strongly convex if α is positive. Consider the primal (P) and the dual (D) standard linear programming problem [2];

$$(P)\begin{cases}\text{Maximize} & \langle s, x_1\rangle \\ \text{subject to} & Ax = k \\ & x_1 \geq 0\end{cases}$$

And the Dual is

$$(D)\begin{cases}\text{Minimize} & \langle k, x_2\rangle \\ \text{subject to} & A^T x_2 \geq s\end{cases} \equiv \begin{cases}\text{minimize} & \langle k, x_2\rangle \\ \text{subject to} & [s - A^T x_2]_+ = 0\end{cases}$$

Consider the equality augmented Lagrangian function given by;

$$l_\alpha(x_1, x_2) = \langle k, x_2\rangle + \langle x_1, [s - A^T x_2]_+\rangle + \frac{1}{2\alpha} \parallel s - A^T x_2 \parallel_2^2$$

and the inequality augmented Lagrangian given by;

$$l_\alpha(x_2, x_1) = \langle k, x_2\rangle + \frac{1}{2\alpha}(\parallel [s - A^T x_2]_+ \parallel_2^2 - \parallel x_1 \parallel_2^2)$$

and consider the penalty function given by;

$$p_\alpha(x_2) = \langle k, x_2\rangle + \frac{1}{2\alpha} \parallel [s - A^T x_2]_+ \parallel^2$$

where $[s - A^T x_2]_+ = max\{s - A^T x_2, 0\}$

That is mean for any number $n \in R$, we lable its non-negative part by $n_+ = max\{n, 0\}$.

2.13 Karush-Kuhu-Tucker (KKT) Conditions

The following four conditions are referred ***KKT*** conditions (for a problem with differentiable (f_j, h_i)) [37];

1-Primal constraints: $f_j(x) \leq 0,\ \&\ h_i(x) = 0,\ i, j = 1, \cdots, \varepsilon.$

2-Complementary slakness: $\beta_j f_j(x) = 0,\ j = 1, \cdots, \varepsilon.$

3-Dual constraints: $\beta \geq 0.$

4-Gradient of Lagrange $w.r.$ to x vanishes:

$$\nabla f_{\circ}(x) + \sum_{j=1}^{\varepsilon} \beta_j \nabla f_j(x) + \sum_{i,j=1}^{\varepsilon} y_j \nabla h_i(x) = 0$$

Example 2.13.1. Consider the problem, find the point for ***KKT*** condition:

$$\begin{cases} \text{maximize} & x_1 + 2x_2 \\ \text{subject to} & x_1 + 3x_2 \leq 5 \\ & 3x_1 + x_2 \leq 7 \\ & x_1 x_2 \geq 0 \end{cases}$$

Solution:.

$f_0(x_1, x_2) = x_1 + 2x_2$

$f_1(x_1, x_2) = x_1 + 3x_2 - 5$

$f_2(x_1, x_2) = 3x_1 + x_2 - 7$

$f_3(x_1, x_2) = -x_1 \leq 0$

$f_4(x_1, x_2) = -x_2 \leq 0$

$$\nabla f_0 = \begin{bmatrix} 1 \\ \\ 2 \end{bmatrix}, \nabla f_1 = \begin{bmatrix} 1 \\ \\ 3 \end{bmatrix}, \nabla f_2 = \begin{bmatrix} 3 \\ \\ 1 \end{bmatrix}, \nabla f_3 = \begin{bmatrix} -1 \\ \\ 0 \end{bmatrix}, \nabla f_4 = \begin{bmatrix} 0 \\ \\ -1 \end{bmatrix}$$

$L((x_1, x_2), (\varphi_1, \varphi_2)) = x_1 + 2x_2 + \varphi_1(x_1 + 3x_2 - 5) + \varphi_2(3x_1 + x_2 - 7)$

1-Primal feasible.

$x_1 + 3x_2 \leq 5$

$3x_1 + x_2 \leq 7 \quad where \quad x_1, x_2 \geq 0$

2-Dual feasible.

$$\begin{bmatrix} 1 \\ \\ 2 \end{bmatrix} - \varphi_1 \begin{bmatrix} 1 \\ \\ 3 \end{bmatrix} - \varphi_2 \begin{bmatrix} 3 \\ \\ 1 \end{bmatrix} - \varphi_3 \begin{bmatrix} -1 \\ \\ 0 \end{bmatrix} - \varphi_4 \begin{bmatrix} 0 \\ \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix}$$

$\varphi_1, \varphi_2, \varphi_3, \varphi_4 \geq 0$

3-Complementary slackness.

$\varphi_1(x_1 + 3x_2 - 5) = 0,\ \ \varphi_2(3x_1 + x_2 - 7) = 0,\ \ \varphi_3(-x_1) = 0,\ \varphi_4(-x_2) = 0.$

4-Stationarity of Lagrangian $= \nabla f((x_1, x_2).(\varphi_1, \varphi_2, \varphi_3, \varphi_4))$

$\varphi_1 + 3\varphi_2 + \varphi_3 = 1$

$3\varphi_1 + \varphi_2 + \varphi_4 = 1$

$\varphi_1, \varphi_2, \varphi_3, \varphi_4 \geq 0$

Since $\varphi_3, \varphi_4 \geq 0$ write the dual feasible

$\varphi_1 + 3\varphi_2 \geq 1$

$3\varphi_1 + \varphi_2 \geq 1$

$\varphi_1, \varphi_2 \geq 0$

Now: find values $x_1, x_2, \varphi_1, \varphi_2, \varphi_3$ and φ_4 the satisfy the ***KKT*** conditions and solve the linear programming problem P. $x_1 + 3x_2 = 5$, $3x_1 + x_2 = 7$, then $\ x_2 = 1, x_1 = 2$.

Must have $\varphi_3 = \varphi_4 = 0$, because $x_1 > 0$ and $\ x_2 > 0$, at optimality and complementary slackness $\varphi_3(-x_1) = 0, \varphi_4(-x_2) = 0$ at optimality. Therefore, $\varphi_1 = \varphi_2 = \dfrac{1}{4}$

The maximum value of the equation is; 5

Definition 2.13.1. ***Mean Square Error (MSE)*** [19]

Mean squared error is a sumation of differences between expected value and true value, if a sample of n data points on all variables produces a vector n predictions, a vector Y is represents the true values of variable being predicted, $\widehat{Y}$ are the expected values (MSR) is calculated as follows;

$$MSR = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2$$

Definition 2.13.2. ***Cost Function*** [19]

It is the variance between expected value and true value to achieve the least error ratio (minumim cost). The cost function is used to calculate the loss based on the expectations made in the linear regression and the Mean Square Error (MSE) is use to calculate the

loss equation is as follows;

$$minimize \frac{1}{m} \sum_{i=1}^{m} (y_{true} - y_{pred})^2$$

where m number of input, y_{true} the real value, y_{pred} the value to be predicted.

Definition 2.13.3. ***Independent variables*** [9]

Independent variables are variables that do not depend on any other variable in the scope of a given experiment, such as time, spase, density and mass, or are variables that are not affected by any other variables called prediction variables independent variables.

Definition 2.13.4. ***Dependent variable*** [9]

A dependent variable is one whose value is determined by an assumption, law or rule depending on the values of other variables and one whose value changes as a result of a change in the independent variable.

CHAPTER 3

NEURAL NETWORKS

3.1 Introduction

A Neural Network is a system with interconnected nodes consisting of several layers of input layer (x), hidden layers and output layer (y), are three layers that make up a neural network. Similar to neurons in the human brain, neural networks use specific algorithms to carry out their responsibilities. These algorithms identify hidden patterns in the data, group and classify them, and as the networks gain experience, their performance gradually improves [33].

Neural networks are known as Artificial Neural Networks (ANNs), Its structure and name were inspired by the human brain because it simulates the communication between neurons and deep learning algorithms are based on this field of machine learning, since the construction of artificial intelligence research was conducted on how to simulate the human mind, and how devices can have the ability to distinguish, how to make a device that distinguishes between right and wrong [4].

As a result of the tremendous progress that the world has witnessed in various fields and activities, it has become necessary to adapt the entries of information systems in these areas [25], since 1950, when the first computer simulation of the biological brain network was introduced, scientific study has focused on the integration of human intelligence into scientific devices [32].

Neural networks are useful in many medical, engineering and communication fields, especially when dealing with large amounts of data. Examples of such applications include the detection of medical phenomena, their monitoring and diagnosis, forecasting the best course of treatment based on the patient's prescription, electrical signal processing, decision-making systems [22], a number of technical analysts used neural networks to predict stock prices in the money market, stock price fluctuations, price indices. Then they compared their forecasts with a number of influencing factors such as past market performance, economic indicators, credit lending [36], in addition to learning and modeling complex and nonlinear relationships between data inputs and

outputs, networks can also find patterns and forecasts, for highly volatile data, such as time series of financial statements, make generalizations and conclusions to reveal hidden relationships between inputs and outputs [4] [52].

Decision-making processes in a variety of agricultural, industrial and engineering fields can also be optimized for all branches, including quality control and processes, credit card fraud detection, personality and voice recognition, computer vision for interpreting primary images and videos such as medical imaging, facial recognition, and others [57].

Researchers are striving for neural networks to fully imitate the human brain and develop into a machine that can perceive and feel its environment, it expect artificial intelligence to be smarter and to be very similar to humans and human intelligence. This has prompted scientists and engineers working on artificial intelligence to try in every way to make it smarter than it is now. Artificial intelligence represents the world around us, conquers all the fields we know, as well as new ones that appear every day [7]

3.1.1 Mathematical Biological Neural Network

The human brain served as the source of inspiration for the neuron's name and design because of how similar it is to the way real neurons interact with one another [35].

A biological neuron starts its mechanism of action when it receives a signal that comes from another cell and the neuron receives that signal through dendrites, when the cell receives the signal, the cell determines whether the signal falls within a certain range, if the signal is within that range, the cell sends a signal to the rest of the cells in the neural network [33]. The neuron functions through a number of fundamental components that cooperate to produce the desired effects [39];

1-Nerve Dendrites: It mostly used to receive inputs in the form of shipments.

2-Cell Body: In order to determine whether the values and charges coming from the dendrites will be passed through the nerve nodes or not, this component processes them.

3-The Nervous Axon: Its purpose is to allow the nerve cell to transmit nerve signals.

4-Synapses of Nerves: This component controls the cell's output and decides how well and how this signal will affect the cell to whom it will be delivered [32]. see figure (3.1)

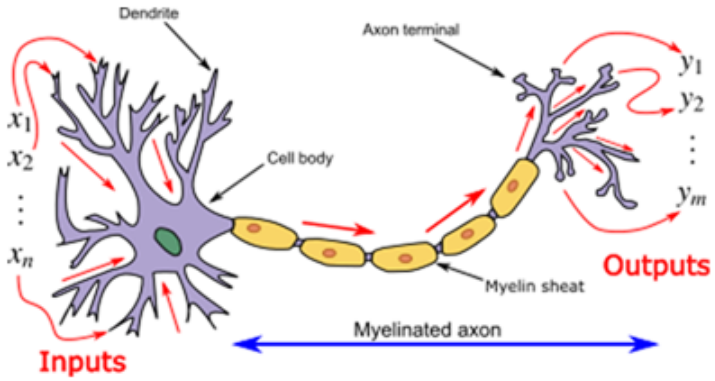


Figure 3.1: A Biological Nerve Cells
The Dendrites represent the input, the Cell body represents the hidden layer in which the calculations are performed and the Nervous Axon represents the output.

3.1.2 Kinds of Neural Networks

Networks typically differ from one another in terms of the quantity, direction, and speed of data [57];

1- Front Feed Networks.

2- Recurrent Neural Networks.

3- Intricate Neural Networks.

4- Self-training Networks.

5- Circular Foundation Networks.

3.2 Neural Network Basics

1-Inputs x **:** It is the values that neurons receive, turning problems and phenomena into a digital form that the network can handle and interact with.

2-Weights w **:** While a neural network function is being used, the weights are multiplied by values to determine their value, the weights are then calculated correctly by applying appropriate mathematical formulas and algorithms during the model-training phase [36]

3-Activation functions σ **:** They are simple to utilize and comprehend due to the fact that they are mathematical functions or equations that enable us to transform activation values to values between $0, 1$.

4-Bias b **:** It is a constant similar to the one found in the equation of a straight line, this value allows us to easily control the results.

5-Outputs y **:** Is the networks ultimate product [4], see figure (3.2)

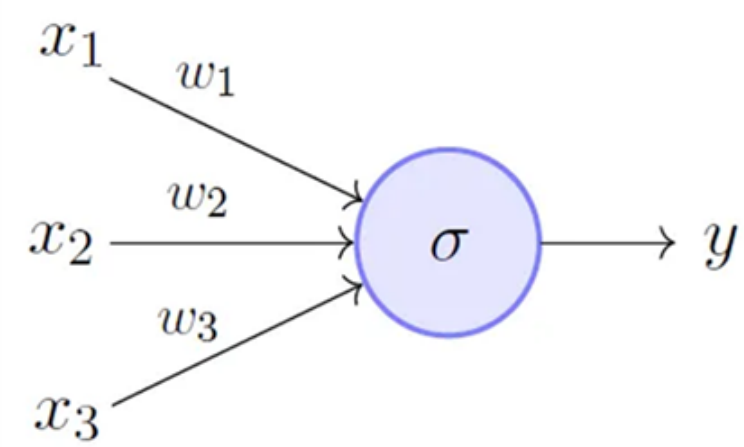


Figure 3.2: Neural Network Basics

3.3 Python

Python is high-level, open source, ready-made and flexible libraries [47]. The popular interpreted Python language can be used to create standalone applications with graphical user interfaces as well as internet applications. Python can be used to write programs directly for new programmers as well as to work on a number of large projects at once [24]. It can also be used as a programming language to manage the functions of

multiple programs. Beginners are often encouraged to learn this language because it is among the fastest educational programming languages [43].

In the second half of the 20th century, Guido van Rossum developed python at the Dutch Institute of mathematics and Informatics in Amsterdam. It premiered in 1991. The programming language C has been used to build the language kernel, some of the most important libraries that have been used include [43];

1-Numpy: Numpy is used to manipulate arrays. Moreover, it has matrices, Fourier transform, and linear algebra functions.

2-Matplotlib: Python is a visualization toolkit for 2D group plots. The ability to visually access huge amounts of data in the form of understandable graphics there are many plots in matplotlib, including Line, Bar, scatter, graph [47].

3-Math: A built-in module called math in the Python standard library provides common mathematical constants and functions. The math module allows users to perform calculations in the areas of numbers, trigonometry, logarithms, and exponentials [24].

3.4 Learning Rate

The learning rate is very important because it represents the core of machine learning applications and determines how quickly the model reaches the point of convergence the point where the error is optimal=0, it is also called the step length.

Therefore, if the learning rate(step length) is large, the steps are fast and less accurate, but if it is small, the steps are slower and more accurate [25], as in the figure(3.3).

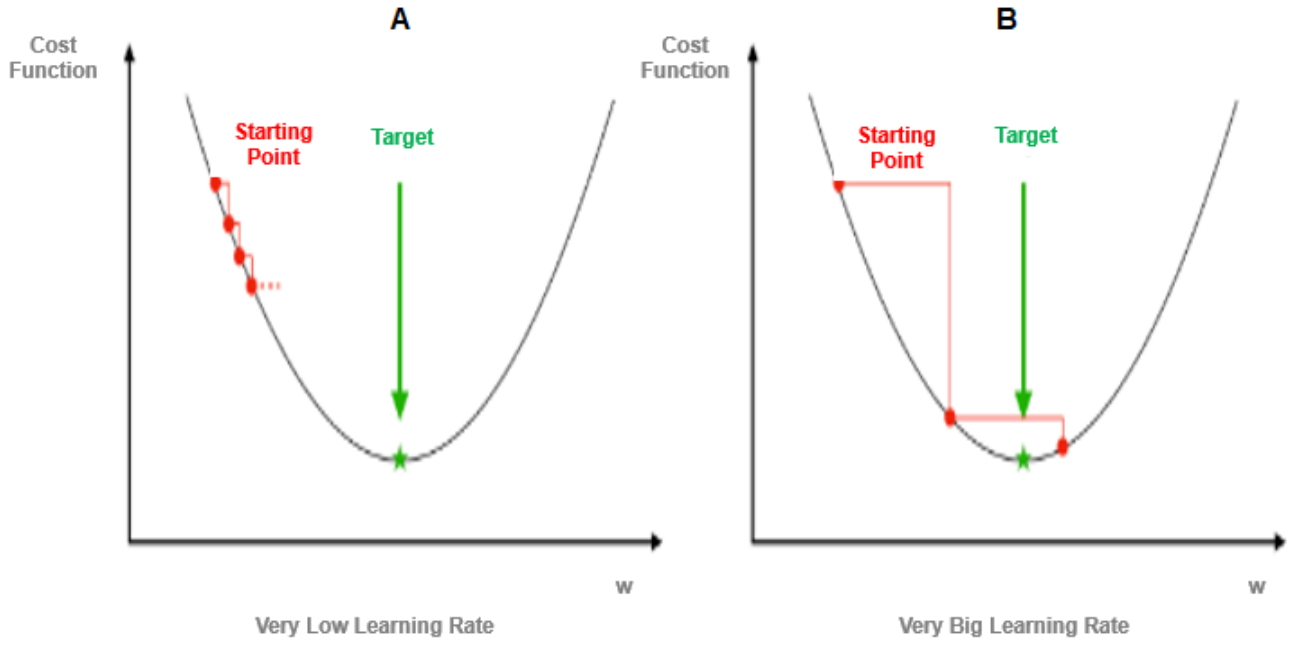


Figure 3.3: Learning Rate
Figure A: when choosing a learning rate that is too small, we need a large number of repetitions to reach the minimum.
Figure B: when choosing a very large learning rate, we need some repetition to reach the minimum, but it may exceed the minimum and lose it.

3.5 Linear Neural Network

The neural network is an interconnected set of nodes or units of basic processing units, approximately analogous to mammalian neurons. The network processing power is retained in the speed of the internet connection or the weights gained during the adaptation (learning) process to the set of training modes, the linear neural network is the basic type of neural network and is used for linear regression, multiple linear regression, polynomial regression and logistic regression [14].

3.6 Linear Regression

The link between one or more independent variables (denoted by x) (input) and a dependent variable (denoted by y) (output) is explained and predicted using regression, all regression issues have a real number ($y \in R$) as their result.
The idea of regression is to have data related to each other and I want to identify new data values that can be applied in stock prices on the stock exchange, house prices, the amount that the client will buy [14].
Must use the following in order to take advantage of regression.

- Features are input data that are either numerical values or vectors.
- There are several pairs (x_i, y_i) in the training examples, which represent the output for each input.
- Feature (model) this describes how the inputs and outputs are related.
- The cost (loss) function or objective function of our model measures its accuracy.
- Reduction of the cost (loss) function or objective function through optimization.

It is also called one-variable regression or (unvariate regression) [29].
Simply put, the set of data to fit in a straight line, where x are the features, y are the goals a supervised learning algorithm (supervised learning) usually works as follows;
The practice set (data) this learning algorithm receives the training set, the task of which is to produce a function $h(x)$, or a hypothesis function, which is subsequently used to generate predictions y for the input of x, see figure (3.4)

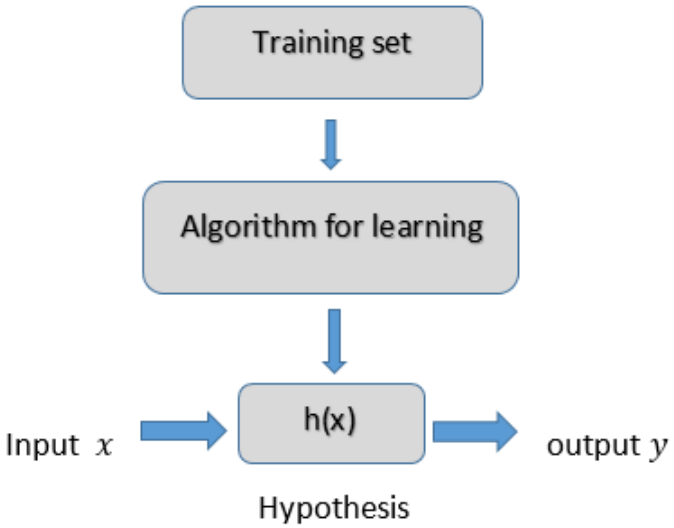


Figure 3.4: Algorithm for Supervised Learning

3.6.1 Linear Equation

Suppose we have the following linear equation.

$$y = mx + c$$

where m : is the slope (or gradeint), c : is the point of intersection with the $y - axis$. We found the slope using two point on the line using the equation [29].

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

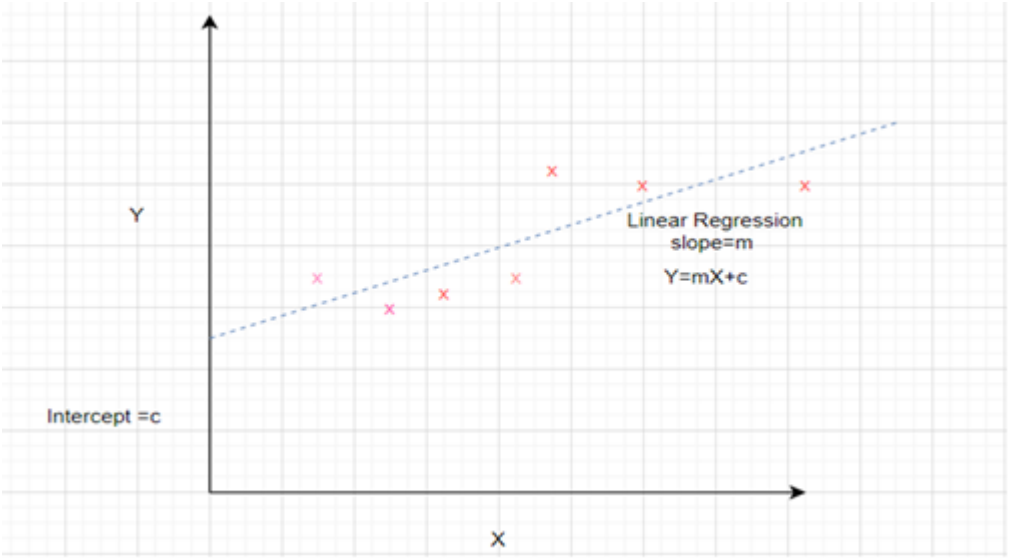


Figure 3.5: Linear Regression and Linear Equation
The dashed line represents a more convenient line and the red dots represent the data distributed on both sides of the line.

A linear equation is an equation whose graph is a straight line, and it is a relationship between two variables that are both of the first order, see figure (3.5). In linear regression many points are given (x_i, y_i), finding a font that complements the identified data the aim (the most suitable font), the most important value in the linear equation is the slope, if the value of the slope is positive, the straight line is upward, if it is negative, the straight line is downward, if the straight line is horizontal, the slope equal zero and if it is vertical, the slope equal infinite (an unknown quantity) [51].

3.6.2 Mathematicals Model of Application Based on Linear Regression

Similar to the equation of a straight line, the linear regression equation can be expressed as follows [19] [25].

Hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters : θ_0, θ_1

Cost function : $j(\theta_0, \theta_1) = \dfrac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$

Where θ_0, θ_1 hypothetical values.

$h_\theta(x_i)$ is the predictable value of linear equation, y_i is the true value.

$$\begin{cases} minimize_{\theta_0,\theta_1} & j(\theta_0, \theta_1) & objective\ function \\ \text{subject to} & \theta_0, \theta_1 \in R & constraint \end{cases} \tag{3.1}$$

Objective is to reduce the variance between the values of $h_\theta(x_i)$ and y_i, and find the value θ_0, θ_1 that make the cost function (cost error function) as low as possible divide by $2m$ to relate the error value to the number of sample values [14].

Example 3.6.1. Let's take the linear equation and the following data.

$h_\theta(x) = \theta_0 + \theta_1 x \Rightarrow h_\theta(x) = 5 + 2x$

Table 3.1: Table data of example (3.6.1)

x	y	$h_\theta(x)$	$h_\theta(x) - y$	$(h_\theta(x) - y)^2$
1	7	7	0	0
2	8	9	1	1
2	7	9	2	4
3	9	11	2	4
4	11	13	2	4
5	10	15	5	25
5	12	15	3	9

Where $\theta_0 = 5,\ \theta_1 = 2,\quad x$: input, $\quad m = 7$ number of input, $\ y$: expected value (output).

$h_\theta(x)$: the true values obtained from the offset of the values of x (input) by the linear equation $h_\theta(x)$.

$h_\theta(x) - y$: the difference between expected value and true value.

$(h_\theta(x) - y)^2$: the difference square of them.

$\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$: sum squared error.

$$j(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

$$j(\theta_0, \theta_1) = \frac{1}{2(7)}(47) = 3.3$$

When choosing random values for θ_0, θ_1 the error value is very large, the error value is too large. θ_0, θ_1 values must be updated continuously to reduce the error. The closer the imposed values are to the real values, the lower the error meaning that we reach the optimal solution, see figure (3.6).

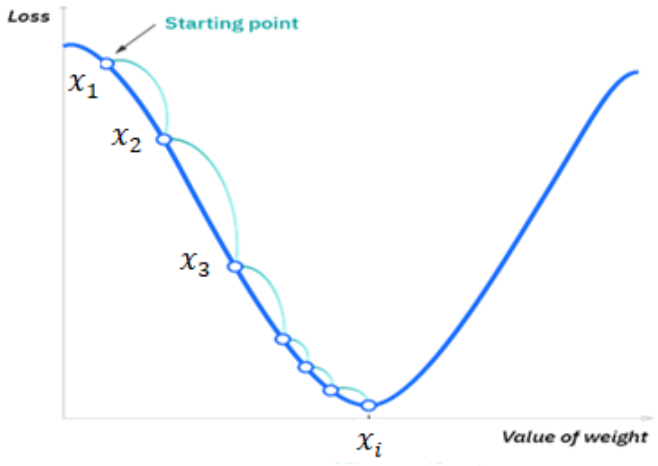


Figure 3.6: The Optimal Solution
$(x_1, x_2, \cdots, x_i)$ Represent the points after ith of the iterations, x_1 starting point, x_i point of convergence, i.e. where the loss function is at its minimum.

The above steps can be applied using the python programming language terms of data represention and drawing the line fit (the most a ppropriate line) when the imposed values are close to the real values [25].

How to choose value θ_0, θ_1 ?

The equation of the straight line $y = mx + c$ is similar to $h_\theta(x) = \theta_0 + \theta_1 x$

Where θ_0 : it represents the point of intersection with $y - axis$. θ_1 : it is slope.

θ_0, θ_1 it determines the most appropriate line (the smaller the differences, the lower the

error rate, and vice versa).

To demonstrate how to choose θ_0, θ_1 Iet's discuss the following examples.

Example 3.6.2. Let $x = 1, 2, 3 \quad y = 1, 2, 3 \quad m = 3$

Table 3.2: Table data of example (3.6.2)

x	y	$h_\theta(x)$	$h_\theta(x) - y$	$(h_\theta(x) - y)^2$
1	1	0	-1	1
2	2	0	-2	4
3	3	0	-3	9

$\theta_0 = \theta_1 = 0 \quad$ then $\quad h_\theta(x) = 0$

$j(\theta_0, \theta_1) = \dfrac{1}{2(3)}(1 + 4 + 9) = \dfrac{14}{6} = 2.33333$

θ_0, θ_1, are small values, but the error value is large. If θ_1 value increases, what happens to cost function ? as in the following example.

Example 3.6.3. Let $\theta_0 = 0 \quad \theta_1 = 0.5 \quad h_\theta(x) = 0.5x \quad m = 3$

$h_\theta(1) = 0.5 \quad h_\theta(2) = 1 \quad h_\theta(3) = 1.5$

Table 3.3: Table data of example (3.6.3)

x	y	$h_\theta(x)$	$h_\theta(x) - y$	$(h_\theta(x) - y)^2$
1	1	0.5	-0.5	0.25
2	2	1	-1	1
3	3	1.5	-1.5	2.25

$j(\theta_0, \theta_1) = \dfrac{1}{2(3)}(0.25 + 1 + 2.25) = 0.5833$

Example 3.6.4. Let $\theta_0 = 0 \quad \theta_1 = 1 \quad h_\theta(x) = x \quad m = 3$

$h_\theta(1) = 1 \quad h_\theta(2) = 2 \quad h_\theta(3) = 3$

Table 3.4: Table data of example (3.6.4)

x	y	$h_\theta(x)$	$h_\theta(x) - y$	$(h_\theta(x) - y)^2$
1	1	1	0	0
2	2	2	0	0
3	3	3	0	0

$j(\theta_0, \theta_1) = 0$

Note that when $\theta_1 = 1$ then the value cost function $= 0$, this mean that the true value applies to the assumed value, and this means the optimal solution, see figure (3.6)

But if the theta value increases, the error value also increases for example.

Example 3.6.5. $\theta_0 = 0 \quad \theta_1 = 1.5 \quad h_\theta(x) = 1.5x \quad m = 3$

$h_\theta(1) = 1.5 \quad h_\theta(2) = 3 \quad h_\theta(3) = 4.5$

Table 3.5: Table data of example (3.6.5)

x	y	$h_\theta(x)$	$h_\theta(x) - y$	$(h_\theta(x) - y)^2$
1	1	1.5	0.5	0.25
2	2	3	1	1
3	3	4.5	1.5	2.25

$$j(\theta_0, \theta_1) = \frac{1}{2(3)}(0.25 + 1 + 2.25) = 0.5833$$

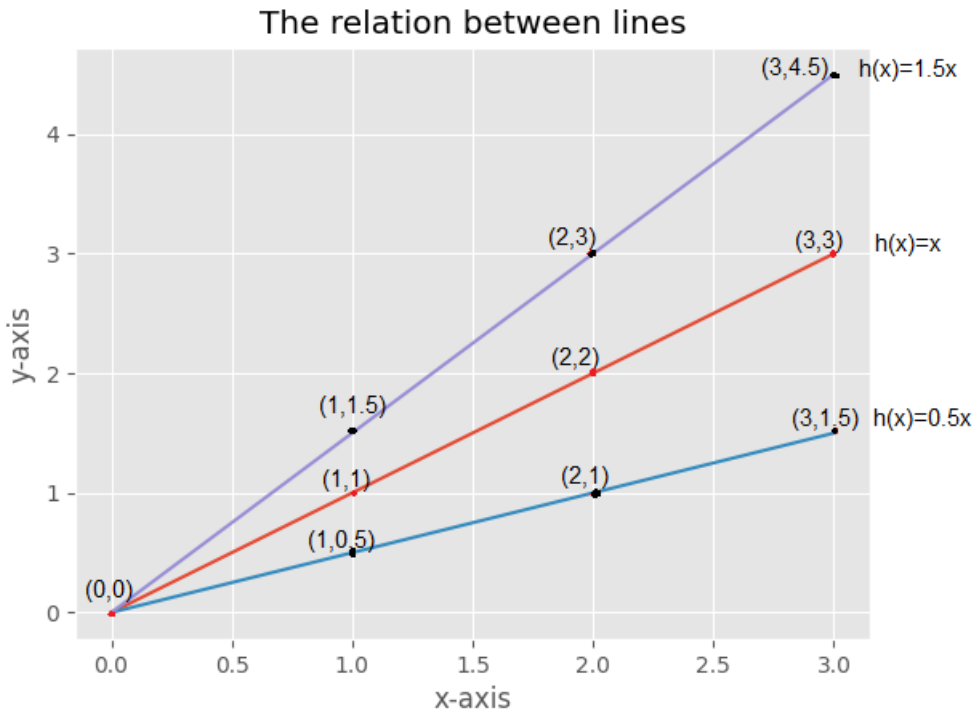


Figure 3.7: The Relation Between Lines

In example (3.6.2) a small θ_0, θ_1 values were chosen after calculating cost function, their value would be large, if θ_1 increased (as in example 3.6.3) the value of error function became less than the previous value, if θ_1 value was updated again and it's value increased (as example 3.6.4) the value of cost function=0, meaning that true value applies to expected value (optimal solution).

In example (3.6.5) large θ_1 value were chosen the value of cost function is also large, so θ_1 values are minimized to get the lowest value of cost function (objective function), see figure (3.7).

3.7 The Gradient Descent Method

The gradient method, which is an iterative procedure for selecting the optimal parameters for reducing the cost function while simulating the relation between a independent variables and dependent variable, to determine a mechanism for reducing the cost function [19].

In order to implement the gradient descent algorithm, we need to reduce the cost function, the number of repetitions, learning rate to calculate the step size at each repetition while advancing towards the minimum, the partial derivatives of theta to update the parameters at each iteration, and the prediction function [30].

After each repetition, the cost is upgraded in proprtion to the error, the error value changes very quickly because it have a square in the error function, so we derive the error function and show it mathematically.

$$
\begin{aligned}
h_\theta(x_i) &= \theta_0 + \theta_1 x_i \\
j(\theta_0, \theta_1) &= \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_i - y_i)^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial j}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i} \left(\frac{1}{2m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x_i) - y_i)^2 \right) \\
&= \frac{1}{2m} \sum_{i=1}^{m} 2 * (\theta_0 + \theta_1 (x_i) - y_i) * x_i^\theta \\
&= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) * x_i^\theta
\end{aligned}
$$

Where

$$
x_i^\theta = \begin{cases} 1 & i = 0 \\ x_i & \text{o.w} \end{cases}
$$

$$\theta_i = \theta_i - \alpha * \left(\frac{\partial}{\partial \theta} cost(\theta_i) \right) = \theta_i - \alpha \frac{1}{m} \left(\sum_{i=1}^{m} (h_\theta(x_i) - y_i) \right) * x_i^\theta \tag{3.2}$$

This update is called the learning rate, which determines the speed at which the model reaches the convergence point, the point at which the error is ideal (as little as possible), see figure (3.8)

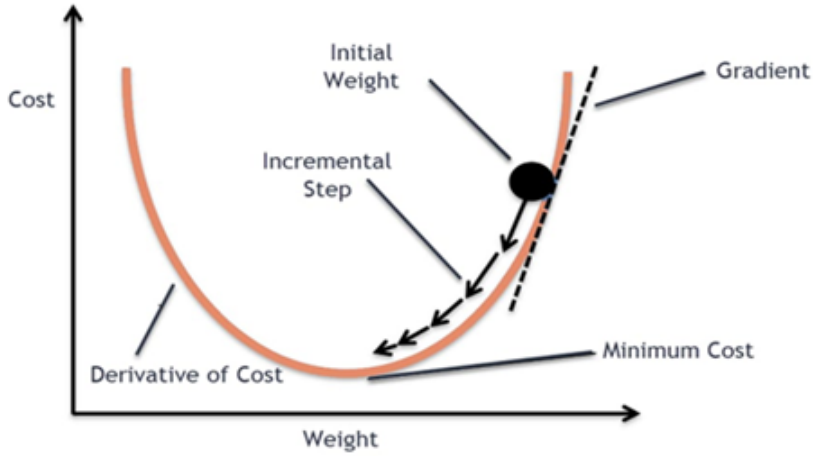


Figure 3.8: Gradient Descent Method
The relationship between the cost function and weight. The alpha value represents the step length, the longer step length, the less accurate the results, but if the step length is small, the more accurate the results.

The above procedures can be used with Python's example (3.6.1) to optimize θ_0, θ_1 values. The results were as follows after selecting the learning rate ($\alpha = 0.01$) and the number of iterations (100), see figure (3.9)(3.10)

Code 1 (Linear Regression)

```
+import . . .
def mean_squared_error(y_true, y_predicted):
    cost = np.sum((y_true - y_predicted) ** 2) / len(y_true)
    return cost
def gradient_descent(x, y, iterations=100, learning_rate=0.01, stopping_threshold=1e-6):
    current_theta1 = 2
    current_theta0 = 5
    iterations = iterations
    learning_rate = learning_rate
    n = float(len(x))
    costs = []
    theta = []
    previous_cost = None
    for i in range(iterations):
        y_predicted = (current_theta1 * x) + current_theta0
        current_cost = mean_squared_error(y, y_predicted)
        if previous_cost and abs(previous_cost - current_cost) <= stopping_threshold:  break
        previous_cost = current_cost
        costs.append(current_cost)
        theta.append(current_theta1)
        theta1_derivative = -(2 / n) * sum(x * (y - y_predicted))
        theta0_derivative = -(2 / n) * sum(y - y_predicted)
        current_theta1 = current_theta1 - (learning_rate * theta1_derivative)
        current_theta0 = current_theta0 - (learning_rate * theta0_derivative)
```

```
        print(f"Iteration {i + 1}: Cost {current_cost}, theta1 \
            {current_theta1}, theta0 {current_theta0}")
    plt.figure(figsize=(6, 4))
    plt.plot(theta, costs)
    plt.scatter(theta, costs, marker='o', color='red')
    plt.title("Cost vs theta")
    plt.ylabel("Cost")
    plt.xlabel("theta")
    plt.show()
    return current_theta1, current_theta0
def main():
    X =np .array([1,2,2,3,4,5,5])
    Y = np.array([7,8,7,9,11,10,12])
    estimated_theta1, estimated_theta0 = gradient_descent(X, Y, iterations=100)
    print(f"Estimated theta1: {estimated_theta1}\nEstimated theta0: {estimated_theta0}")
    Y_pred = estimated_theta1 * X + estimated_theta0
    plt.figure(figsize=(6, 4))
    plt.scatter(X, Y, marker='o', color='red')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='blue', markerfacecolor='red',
        markersize=10, linestyle='dashed')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()
```

Out···

$Estimated\ \theta_0 : 5.035844310464995$ $Estimated\ \theta_1 : 1.2783066626429578$

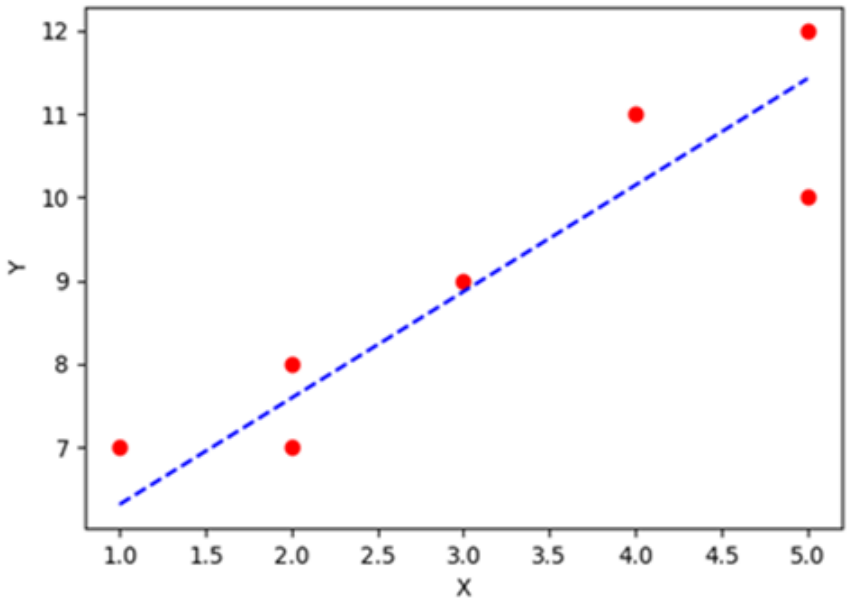


Figure 3.9: Best Fit Line
The dashed line shows the most convenient line and the data (red dots) are distributed on both sides of the line, the presence of values close to the line means that the error rate is low and distant values mean that the error rate is large.

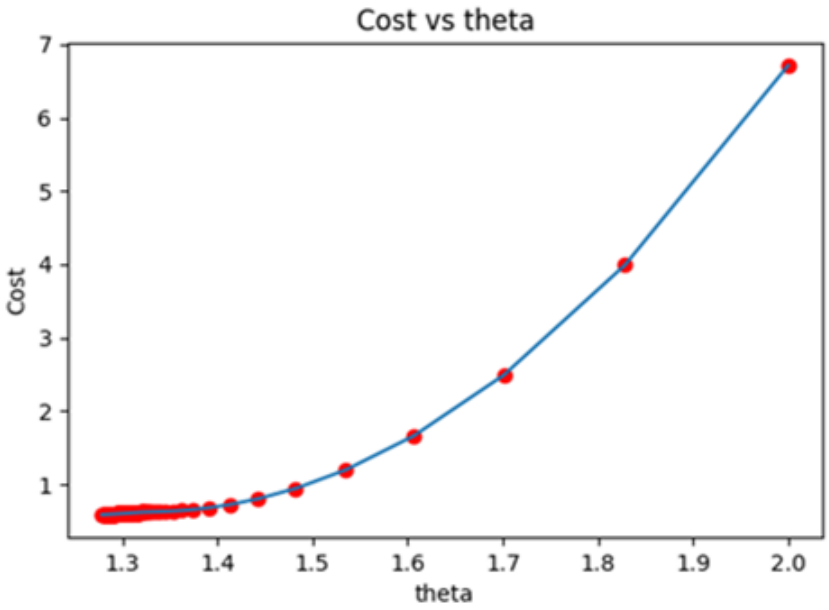


Figure 3.10: Relation Between Cost and Theta
Update in the value of the theta parameter(decreases) reduces the cost function.

3.8 Multiple Linear Regression

The expectation of one variable (input x and output y) now we are dealing with more than one variable in the sense that the internal data has more than one information, instead of entering the area of the house to find out its price, we enter the area of the house, its location, the number of rooms, its age and others to determine its price, these inputs are called features. Each independent variable in this situation has an effect on the expected result [14].

For each entry, the corresponding Theta that informs the model will be determined based on the data points that best indicate the importance of each entry, the formula is as follows.

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$

where $x_0 = 1 \quad x, \theta$ are vectors $x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \in R^{n+1} \quad \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{n+1}$

The above equation can be written as follows. $h_\theta(x_i) = \theta_j^T x_i = \begin{bmatrix} \theta_0 & \cdots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_1 \end{bmatrix}$

where $i = 1, \cdots, m \quad j = 0, \cdots, n \quad \theta_i^T$ is transpose (is obtained by interchanging the row and column).

Cost function $(loss) : j(\theta_j) = \dfrac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$

Repeat until convergence :

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) * x_i \tag{3.3}$$

Example 3.8.1. Assume that we have number of goods and each commodity it has three features x_1, x_2, x_3, $\theta_0 = 5,\ \theta_1 = 2,\ \theta_2 = 3,\ \theta_3 = 4, \quad m = 5 \quad n = 4$ as shown in the table.

Table 3.6: Table data of example (3.8.1)

x_0	x_1	x_2	x_3	y	$h(x)$	$h(x) - y$
1	3	20	13	120	123	3
1	6	16	11	113	109	-4
1	5	12	8	85	83	-2
1	7	17	9	100	106	6
1	4	10	7	60	71	11

We $n + 1$ features because that are $x_0 = 1, \quad \sum_i (h(x) - y) = 14$

$$x_1 = \begin{bmatrix} 1 \\ 3 \\ 20 \\ 13 \end{bmatrix}, x_2 = \begin{bmatrix} 1 \\ 6 \\ 16 \\ 11 \end{bmatrix}, x_3 = \begin{bmatrix} 1 \\ 5 \\ 12 \\ 8 \end{bmatrix}, x_4 = \begin{bmatrix} 1 \\ 7 \\ 17 \\ 9 \end{bmatrix}, x_5 = \begin{bmatrix} 1 \\ 4 \\ 10 \\ 7 \end{bmatrix}$$

$$h(x_i) = \theta_j^T * x_i \Rightarrow h(x_1) = \begin{bmatrix} 5 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 20 \\ 13 \end{bmatrix} = [5 + 6 + 60 + 52] = 123$$

$h(x_2) = 109 \qquad h(x_3) = 83 \qquad h(x_4) = 106 \qquad h(x_5) = 71$

Repeat until convergence: suppose $\alpha = 0.01, \quad m = 5$

$$\theta_0 = 5 - \frac{0.01}{5}(14) * 1 = 4.972, \quad \theta_1 = 2 - \frac{0.01}{5}(14) * 3 = 1.916$$

$$\theta_2 = 3 - \frac{0.01}{5}(14) * 20 = 2.44, \quad \theta_3 = 4 - \frac{0.01}{5}(14) * 13 = 3.636$$

Theta values are starting to decrease, we report the operation several times until we reach the optimal value, and can be used python.

Now; What is the right number for the number of attempts ?

The following figure (3.11) shows the relationship between the number of attempts (iterations) and the reduction of the cost function.

It's clear from the drwing that the more the number of attempts, the lower cost function, but after a certain period the tendency approaches zero, so the number of attempts becomes large with a slight change in the value of the cost function.

So, we must stop because it is considered a wast of time and effort [28].

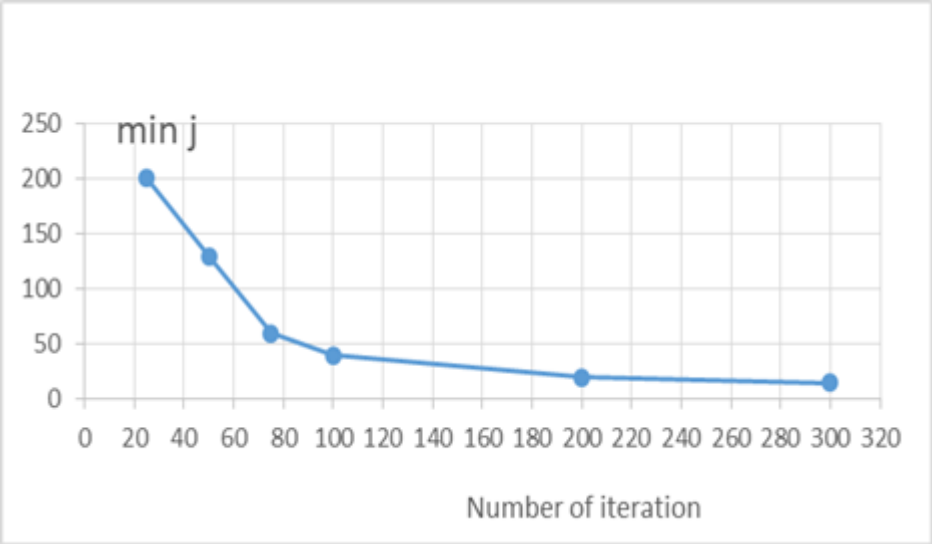


Figure 3.11: Relation Between Min j and Number of Iteration

Can stop after (100 itrations) or more attempts, each case is different from the others, choice of the alpha value affects the number of attempts and the accuracy of the data, so the number of attempts and speed should be appropriate to discover the best solution.

3.9 Normal Equation

Normal Equation is an analytical approach used for optimization an alternative to linear regression, the equation is reduced without repetition, this approach is an effective approach time-saving option [30].

The normal equation method is based on the mathematical concept of small and large values, where the partial derivative of any function is zero at the minimum and maximum. It can be said that equations obtained by determining the partial derivatives of the sum of squared errors or the cost function equal to zero, where a person can estimate the coefficients of multiple linear regression [19].

$h(\theta) = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n$

$h(\theta) = \theta^T x$

$j(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$

where m: features, $i = 1, ..., m$, x_i: input value, y_i: expected value, represent the cost function as vector, $x_0 = 1$

$$\begin{bmatrix} h_\theta(x_0) \\ \vdots \\ h_\theta(x_m) \end{bmatrix} - \begin{bmatrix} y_0 \\ \vdots \\ y_m \end{bmatrix}$$

$\frac{1}{2m}$ was ignored because it doesn't make any difference in the work. it was used mathematically during the calculation of gradient descent.

$$\begin{bmatrix} \theta^T(x_0) \\ \vdots \\ \theta^T(x_m) \end{bmatrix} - y \Rightarrow \begin{bmatrix} \theta_0(x_{00}) + \theta_1(x_{01}) + \cdots + \theta_n(x_{0n}) \\ \vdots \\ \theta_n(x_{m0}) + \theta_1(x_{m1}) + \cdots + \theta_n(x_{mn}) \end{bmatrix} - y$$

It can be written as $[X\theta - y]$ since the formula a bove can not squared since the square of the vector (matrix) is not equal to the square of each of it's values to obtain the squared value the vector can be multiplied by transpose.

Therefore, the Cost Function is: $j(\theta) = (X\theta - y)^T (X\theta - y)$

$$\begin{aligned}
\frac{\partial j(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta}[(X\theta - y)^T (X\theta - y)] \\
&= 2X^T X\theta - 2X^T y \qquad cost'(\theta) = 0 \\
0 &= 2X^T X\theta - 2X^T y \\
2X^T X\theta &= 2X^T y \\
(X^T X)^{-1}(X^T X)\theta &= (X^T X)^{-1}(X^T y) \\
\theta &= (X^T X)^{-1}(X^T y) \\
where \quad (X^T X)^{-1} &\neq (X^T X) \\
(X^T X)^{-1}(X^T X) &= I \qquad (identity\ matrix)
\end{aligned} \tag{3.4}$$

This is the normal equation with θ giving the minimum cost value [30].

Example 3.9.1. solving the previous example (3.8.1) by the Normal equation;

$$X = \begin{bmatrix} 1 & 3 & 20 & 13 \\ 1 & 6 & 16 & 11 \\ 1 & 5 & 12 & 8 \\ 1 & 7 & 17 & 9 \\ 1 & 4 & 10 & 7 \end{bmatrix} \qquad X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 6 & 5 & 7 & 4 \\ 20 & 16 & 12 & 17 & 10 \\ 13 & 11 & 8 & 9 & 7 \end{bmatrix}$$

$$X^T = \begin{bmatrix} 5 & 25 & 75 & 48 \\ 25 & 135 & 375 & 236 \\ 75 & 375 & 1189 & 755 \\ 48 & 236 & 755 & 484 \end{bmatrix} \qquad (X^TX)^{-1} = \begin{bmatrix} \frac{85841}{7870} & \frac{-8283}{7870} & \frac{410}{787} & \frac{-1087}{787} \\ \frac{-8283}{7870} & \frac{1299}{7870} & \frac{-70}{787} & \frac{128}{787} \\ \frac{410}{787} & \frac{-70}{787} & \frac{108}{787} & \frac{-175}{787} \\ \frac{-1087}{787} & \frac{128}{787} & \frac{-175}{787} & \frac{320}{787} \end{bmatrix}$$

$$(X^TX)^{-1}.X^T = \begin{bmatrix} \frac{841}{3935} & \frac{-17827}{7870} & \frac{3333}{3935} & \frac{-27}{787} & \frac{17619}{7870} \\ \frac{-873}{3945} & \frac{2391}{7870} & \frac{26}{3935} & \frac{43}{787} & \frac{-1127}{7870} \\ \frac{85}{787} & \frac{-207}{787} & \frac{-44}{787} & \frac{181}{787} & \frac{-15}{787} \\ \frac{-43}{787} & \frac{401}{787} & \frac{13}{787} & \frac{-286}{787} & \frac{-85}{787} \end{bmatrix}$$

$$(X^TX)^{-1}.X^Ty = \begin{bmatrix} -27.42833545 \\ 5.141423126 \\ 0.3418043202 \\ 9.603557814 \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Can be use python(2)

Code 2 (Normal equation)

```
   +import . . .
x1=([3,6,5,7,4])
 x2=([20,16,12,17,10])
x3=([13,11,8,9,7])
y=([120,113,85,100,60])
print(x1)
print(x2)
print(x3)
plt.scatter (x1,y)
plt.scatter(x2,y)
plt.scatter (x3,y)
plt.scatter (x3,y)
plt.title ('Normal equation')
plt.xlabel ('x_axis')
plt.ylabel ('y_axis')
plt.show()
x1=np.array(x1)
x2=np.array(x2)
x3=np.array(x3)
y=np.array(y)
n=len(x1)
print('n=',n)
x_bais=np.ones((n,1))
x1_new=np.reshape(x1,(n,1))
x2_new=np.reshape (x2,(n,1))
x3_new=np.reshape (x3,(n,1))
```

```
x_new=np.append (x_bais ,x1_new ,axis=1)
x_new=np.append (x_new ,x2_new,axis=1)
x_new=np.append (x_new ,x3_new ,axis=1)
print('x_new=',x_new )
x_new_transpose=np.transpose (x_new)
x_new_transpose_dot_x_new=x_new_transpose.dot(x_new)
temp_1=np.linalg.inv(x_new_transpose_dot_x_new )
temp_2=x_new_transpose .dot(y)
theta=temp_1.dot(temp_2)
print('theta=',theta)
theta_0=theta[0]
theta_1=theta[1]
theta_2=theta[2]
theta_3=theta[3]
print('theta0=',theta_0)
print('theta1=',theta_1)
print('theta2=',theta_2)
print('theta3=',theta_3)
def predict_values(theta_0,theta_1,theta_2,theta_3,x1,x2,x3):
    predicted_value=(theta_0+theta_1*x1 +
                theta_2*x2+theta_3*x3)
    return predicted_value
print('predict_values=',predict_values)
```

Out···

$$[3, 6, 5, 7, 4], [20, 16, 12, 17, 10], [13, 11, 8, 9, 7]$$

$x_{new} = [[1.3.20.13.][1.6.16.11.][1.5.12.8.][1.7.17.9.][1.4.10.7.]]$

$theta = [-27.42833545 \quad 5.14142313 \quad 0.34180432 \quad 9.60355781]$

$theta_0 = -27.428335451078965 \qquad theta_1 = 5.1414231257945175$

$theta_2 = 0.3418043202034369 \qquad theta_3 = 9.603557814485612$

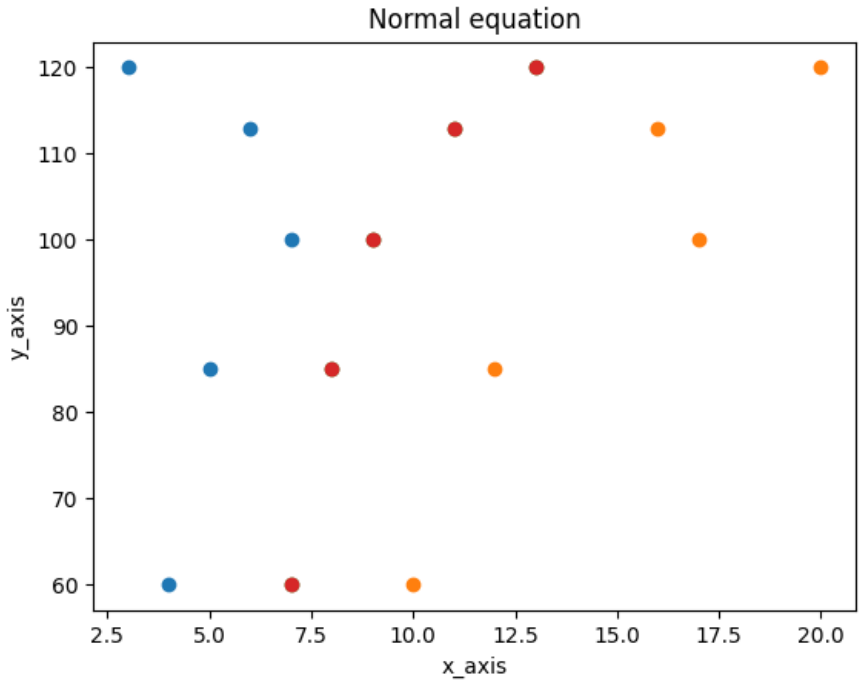


Figure 3.12: Data Representation Space
Points blue refer to $x_1 = [3, 6, 5, 7, 4]$, points orange refer to $x_2 = [20, 16, 12, 17, 10]$, points red refer to $x_3 = [13, 11, 8, 9, 7]$, with $y = [120, 113, 85, 100, 60]$.

3.9.1 The Distinction Between the Gradient Descent and Normal Equation

Gradient descent	**Normal equation**
1-Be slow and need to choose learning rate.	1-Be fast and not need to choose learning rate.
2-It's an iterative algorithm.	2-It's an analytical approach.
3-It works fairly well with a lot features.	3-It works fairly well with a few features.
4-You can use the scaling feature.	4-Does not use the scaling feature.

Sometimes a problem occurs when applying the normal equation because the matrix is not invertible or it is singular, it has no inverse, or the number of rows is less than the number of columns, so we need to increase the number of rows or reduce the number of columns, or there are data, one of which is dependent on the other this leads to the determinant of the matrix =0 (it has no inverse) [21].

3.10 Polynomal Regression

linear regression, can't apply as a one-size fits all answer to every issue. Numerous interactions between variables in the actual world are non-linear, meaning that a straight line cannot accurately depict the relationship.

For these issues, we employ polynomial regression, an extension of the prior linear regression that can account for additional complexity, such as curves. This technique looks for non-linear issues by applying various powers to the explanatory variable [14].

In summary. In order to fit the data linearly, linear regression is a linear model; which is unable to fit and identify patterns in non-linear data. Thus, it determines the degree to which the independent variables (features) and the dependent variable(target variable) are related [35], the following form of the equation can be used.

$$y = \theta_0 x^0 + \theta_1 x^1 + \cdots + \theta_n x^n$$

The following formula can be used to represent the equation:

$$y = \theta_0 + \sum_{i=1}^{n} \theta_i x^n, \quad where \quad i = 1, \cdots, n. \tag{3.5}$$

A model like this can yield a second order, third order, or *nth* order equation that fits the data points in addition to capturing a straight line (if necessary) from the prior equation.

Example 3.10.1. Consider the following data house prices by specific region, find the expected values of house prices.

x	1	2	3	4	5	6	7
y	44000	50000	59000	79000	115000	150000	180000

To understand why polynomial regression is needed, linear regression was first applied to non-linear data. Python can be used to solve the example (3.10.1).

Code 3 (Linear Regression and Polynomial Regression)

```
+import . . .
def main():
   X = np.array([[1],[2], [3], [4], [5], [6], [7]])
   Y = np.array([44000, 50000, 59000, 79000, 115000, 150000, 180000])
   print('X=',X)
   print('Y=',Y)
   model = LinearRegression()
   model.fit(X,Y)
   Y_pred1 = model.predict(X)
   print('Y_pred1=',Y_pred1 )
  plt.scatter(X, Y, color='blue')
   plt.plot(X, Y_pred1, color='red')
   plt.title('(Linear Regression)')
   plt.xlabel('X-axis')
   plt.ylabel('Y-axis')
   plt.show()
```

```
if __name__ == "__main__":
    main()
class PolynomailRegression():
    def __init__(self, degree, learning_rate, iterations):
        self.degree = degree
        self.learning_rate = learning_rate
        self.iterations = iterations
    def transform(self, X):
        X_transform = np.ones((self.m, 1))
        j = 0
        for j in range(self.degree + 1):
            if j != 0:
                x_pow = np.power(X, j)
                X_transform = np.append(X_transform, x_pow.reshape(-1, 1), axis=1)
        return X_transform
    def normalize(self, X):
        X[:, 1:] = (X[:, 1:] - np.mean(X[:, 1:], axis=0)) / np.std(X[:, 1:], axis=0)
        return X
     def fit(self, X, Y):
        self.X = X
        self.Y = Y
        self.m, self.n = self.X.shape
        self.W = np.zeros(self.degree + 1)
        X_transform = self.transform(self.X)
         X_normalize = self.normalize(X_transform)
        for i in range(self.iterations):
            h = self.predict(self.X)
```

```
            error = h - self.Y
            self.W = self.W - self.learning_rate * (1 / self.m) * np.dot(X_normalize.T, error)
        return self
    def predict(self, X):
        X_transform = self.transform(X)
        X_normalize = self.normalize(X_transform)
        return np.dot(X_transform, self.W)
def main():
    X = np.array([[1],[2],[3],[4],[5],[6],[7]])
    Y = np.array([44000, 50000, 59000, 79000, 115000, 150000, 180000])
    model = PolynomailRegression(degree=2, learning_rate=0.01, iterations=500)
    model.fit(X, Y)
    Y_pred2= model.predict(X)
    print('Y_pred2=',Y_pred2)
    plt.scatter(X, Y, color='blue')
    plt.plot(X, Y_pred2, color='red')
    plt.title('(Polynomial Regression)')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.show()
if __name__ == "__main__":
    main()
```

Out (1) $\cdots$

$X = [[1][2][3][4][5][6][7]]$

$Y = [44000 \quad 50000 \quad 59000 \quad 79000 \quad 115000 \quad 150000 \quad 180000]$

$Y_{pred1} = [25571.42857143 \quad 49285.71428571 \quad 73000. \quad 96714.28571429 \quad 120428.57142857$

$144142.85714286 \quad 167857.14285714]$

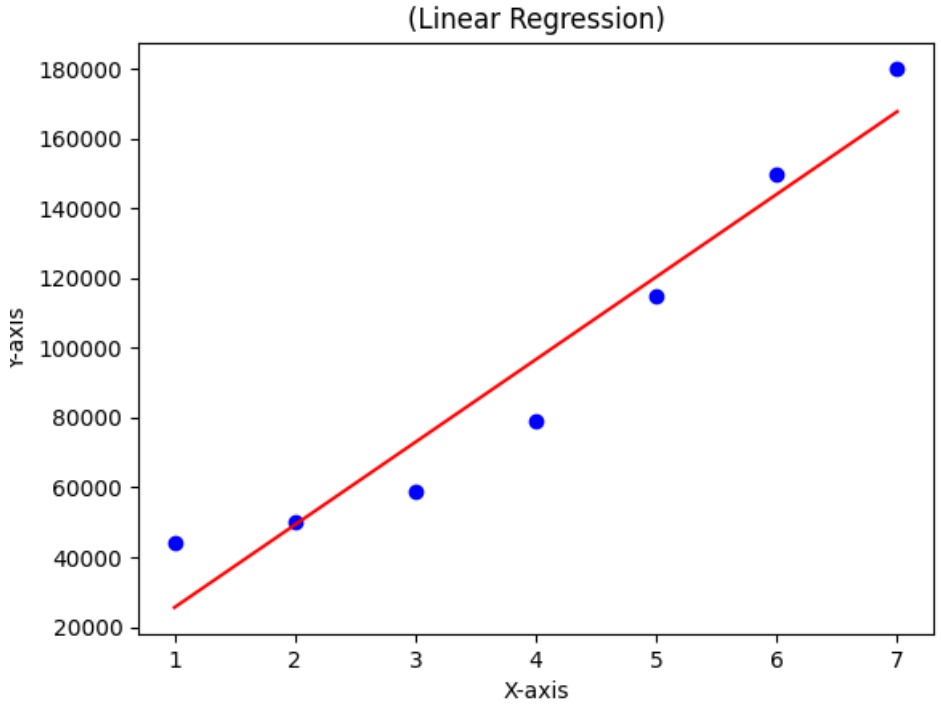


Figure 3.13: Linear Regression
The red line represents the most appropriate line and the blue dots represent the data distributed on both sides of the line.

By use equation of linear regression (3.3) its default function is linear in nature and the training data is a non-linear of X in the picture, linear regression is not even able to match the training data adequately, figure (3.13). In addition, the forecast data are very far from the real values.

Out (2) $\cdots$

$Y_{pred2} = [32444.42457929 \quad 48033.28407594 \quad 66995.70818674 \quad 89331.69691171$

$115041.25025085 \quad 144124.36820414 \quad 176581.0507716]$

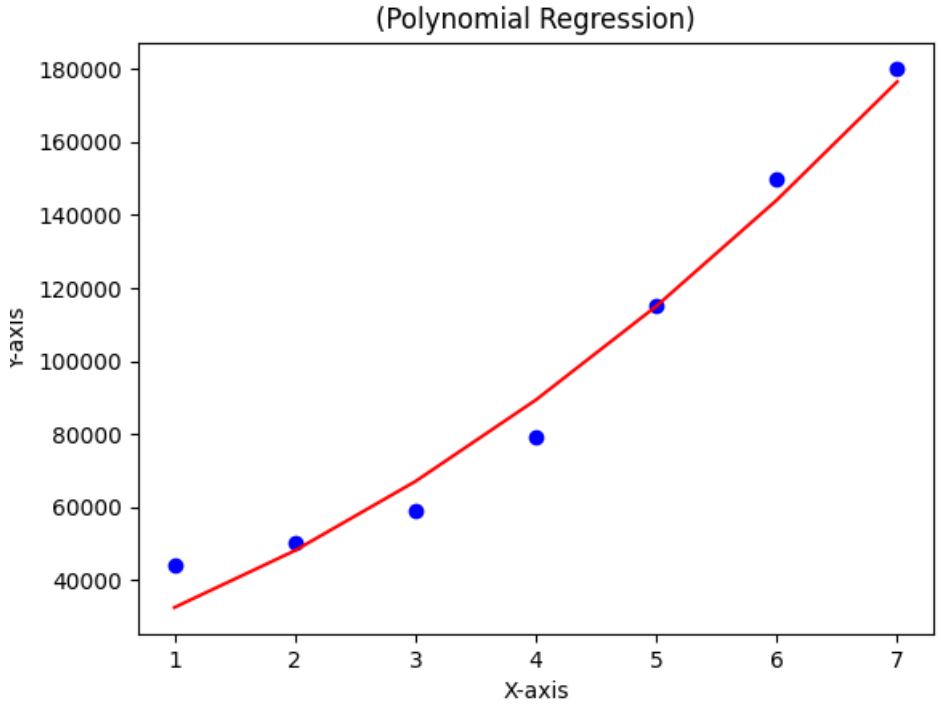


Figure 3.14: Polynimial Regression
The red curve represents the most favorable curve for the data and the blue dots represent the data distributed on the sides.

By use equation of polynomial regression (3.5) the output(2) picture demonstrated how polynomial regression produced a curve to fit the nonlinear data, see figure (3.14). In addition, the forecast data is very close to the real values.

3.11 Logistic Regression

Logistic regression is a moderate model where values between (1 or 0) are predicted, a change in any coefficient leads to a change in the direction and intensity of the logistic function, positive slopes lead to a curve resembling the letter S, negative slopes produce a curve resembling the letter Z. Activation functions are used to convert linear regression to logistic, each time the conversion the accuracy is used to predict of the weights. [14].

3.12 Activation Functions

Neural networks consist of very important elements, specifically, the output, hidden layers and input layer. The input layer receives the raw data from the domain no calculations are made here ; the data (features) are simply sent to the hidden layer, the output layer then receives any calculations made on the input data by the hidden layer and then this layer of the network gives the last step to the values, the two main terms used to describe the flow of information during the training of neural networks are forward and backward propagation, the activation function acts as a mathematical link between the inputs feeding the current neurons and their outputs, which are sent to the next layer in the forward feed, in the backward propagation the weights are repeatedly adjusted to minimize discrepancies between the actual output vector of the network and the desired output vector [40].

There are several distinct types of activation functions and they can be divided into linear and non-linear forms. This property is very important, especially when training neural networks. The most important goal in activation functions is to give nonlinearity to neural networks among the main ones, we list the following [14].

3.12.1 Sigmoid Function

This function accepts any real value as an input for this function, which returns values between(0 and 1). The output value will be nearer to 1 (more positive) the greater the input, and nearer to 0 the smaller the input (more negative) [40], since the chance of any value occurring is only between the ranges of (0 and 1), this function is said to be the most common. It is also him gives a smooth gradient because it's derivable, which prevents jumps in output numbers. It can be represented mathematically as follows;

$$f(x) = \frac{1}{1 + e^{-x}}$$

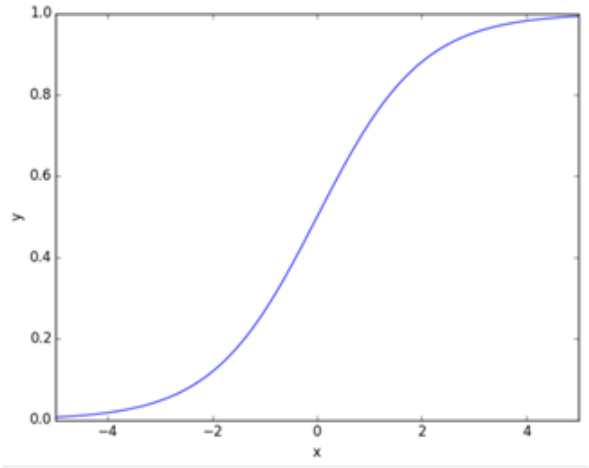


Figure 3.15: Sigmoid Activation Function

The derivative of the sigmoid function is as follows;

$$f'(x) = sigmoid(x) * (1 - sigmoid(x)) = \frac{e^{-x}}{(1 - e^{-x})^2}$$

3.12.2 Tanh Function

Tanh function has the same structure as the sigmoid function and is quite comparable to it, but the output values range from (-1) to (1), depending on the amount of positive or negative input. If the input is more positive, the output tends towards (1), while if it is more negative, it tends towards (-1) [14].

$$f(x) = tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

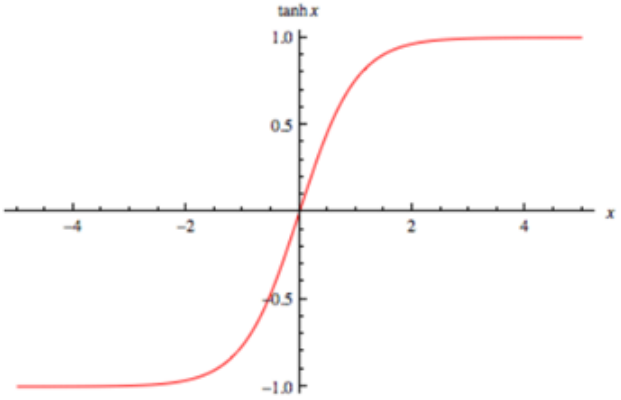


Figure 3.16: Tanh Activation Function

It is continuous and derivable and its derivative is;

$$f'(x) = 1 - tanh^2(x)$$

3.12.3 ReLU Function

Rectified linear unit (ReLU) does not activate all neurons at once, while giving the idea that a linear function has a derivative and allowing backpropagation, which makes it computationally efficient [33], can be written mathematically;

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

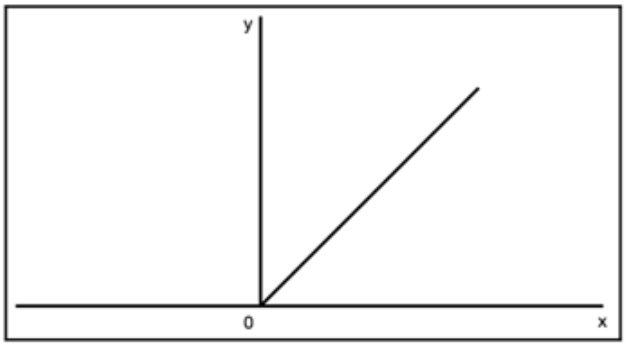


Figure 3.17: ReLU Activation Function

And the derivative function has;

$$f'(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

3.12.4 Leaky ReLU

An enhanced version of the ReLU function that tackles the problem of dying really is the ReLU leaky function. The color gamut on the left side of the graph is no longer 0 when the values of negative inputs are modified, resolving the problem of dying neurons in that area. It also has a little positive slope in the negative area [8], can be written mathematically;

$$f(x) = \begin{cases} x & if\ x > 0 \\ 0.01 & if\ o.w \end{cases}$$

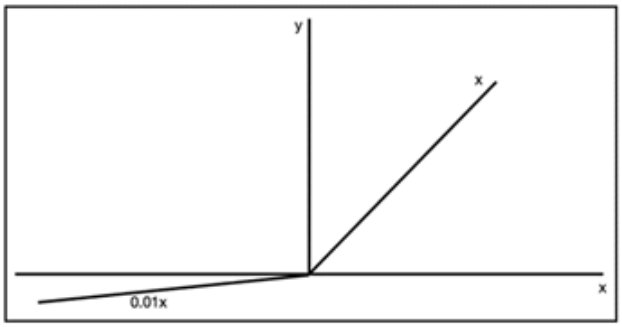


Figure 3.18: Leaky ReLU Activation Function

And the derivative function has;

$$f'(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0.01 & if\ x < 0 \end{cases}$$

3.12.5 Parametric ReLU Function

Another variation of relais ReLU parametric, where the left half of the axis is set to zero to solve the gradient problem. Backpropagation is used to determine the most appropriate value of (0), which is the slope of the function's negative component when the leaky function is still unable to address the issue of dead neurons and the necessary information is not effectively transferred to the following layer, the ReLU parameter function is utilized, can be written mathematically [14];

$$f(x) = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ o.w \end{cases}$$

where α is the slope parameter for negative value.

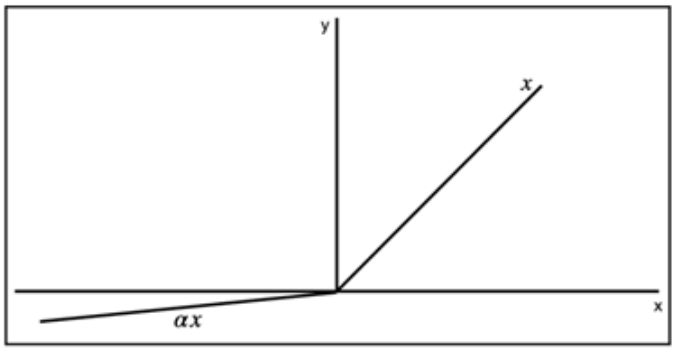


Figure 3.19: Parametric ReLU

And the derivative function has;

$$f'(x) = \begin{cases} 1 & if\ x < 0 \\ \alpha & if\ o.w \end{cases}$$

CHAPTER 4

ALGORITHMS OF NEURAL NETWORKS

4.1 Introduction

The basic building block for creating a neural network is the neuron. The neural network takes the input, multiplies the input values with the relevant weights and generates an output. A type of machine learning called neural networks uses huge amounts of information for adaptation and learning (data), and each node in the neural network consists of two functions in forward propagation: the linear function and the activation function [28]. These two functions are used to represent hidden layers and outputs by multiplying the nodes that were previously connected by their product multiplier, together with the relevant weight and bias. Following the application of the linear function and activation functions as ReLU, parametric ReLU, sigmoid, leaky ReLU, etc. are carried out according to the specifications and the nature of the problem [36].

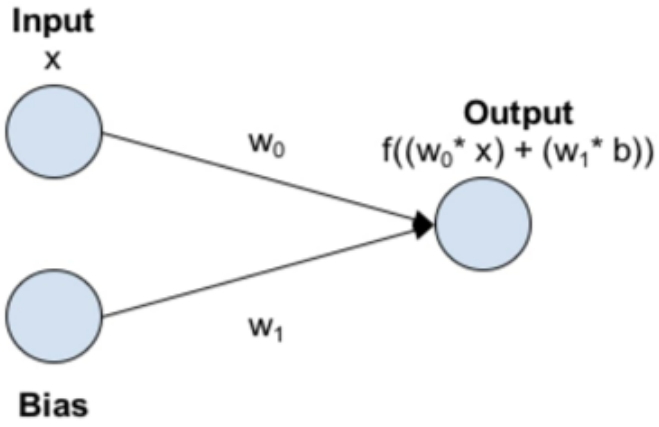


Figure 4.1: Forward Propagation Algorithm

Neural network algorithms are techniques or series of logical and mathematical procedures that must be taken in order to solve an issue neural network algorithm is created by studying and analyzing events that have already occurred and are recorded

in previous records in databases, comparing the predictions made by the neural network with previous real records, and then updating the weights with more accurate values [6]. Weights are considered the most important in the construction of a neural network because they use a good and accurate prediction, techniques that alter the learning rate and weights of neural network parameters to make them more cost-effective function optimizers employ objective function reduction (loss) to resolve optimization problems [57]. After computing the output in the output layer, the cost function is used to compare and calculate the estimated (expected) and real (actual) value, where it is chosen how to alter the neural network weights or learning rates. Then, in order to assess the experiment's outcome from the actual output, optimization algorithms are employed to calculate the values of new weights, or the change of weight loss [4].
This chapter will outline various optimization algorithms and explain how each one reduces the cost function.

4.1.1 Algorithm of Gradient Descent

Gradient descent is an optimization approach for determining the local minimum of a derivable (differential) function, which is based on a convex function and repeatedly changes its parameters to reduce the cost function to a local minimum. Gradient descent is used to train neural network models [6], when(point $x^\star \in F(F$ is a feasible set) is a local minimum if there is an open neighborhood N of $x^\star$ such that $f(x^\star) \leq f(x), for\ x \in N$).
Since the first-order derivative of the objective function, reflecting the slope or rate of change, is used in the gradient descent procedure, it is a first-order optimization algorithm. Multivariate function is an objective function that accepts a lot of different variables [17]. A vector can be used to represent the input variables, and a vector can also be used to represent the derivative of a multivariate function, also called (gradient where the gradient of $j(\theta)$ is the n-dimensional vector of paratial derivatives of j, and

denoted by $\nabla j(\theta)$), so the gradient descent algorithm requires a function to be improved $j(\theta)$ and a derivative function of the objective function $\nabla j(\theta)$ [21].

$$\nabla j(\theta_i) = \begin{bmatrix} \frac{\partial h(x)}{\partial \theta_1} \\ \vdots \\ \frac{\partial h(x)}{\partial \theta_i} \end{bmatrix}, \ where \ i = 1, \cdots, n$$

this algorithm is used to determine the values of the parameters that lower the cost function as much as possible by first defining values for the starting parameter and repeatedly adjusting the values using the gradient descent technique (differential calculation) [3].

The weight is configured using some algorithms and updated at each step of it according to the update equations as follows.

Hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters : θ_0, θ_1

Cost function : $j(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$

$$\begin{cases} \text{Minimize} & j(\theta_0, \theta_1) \\ \text{Subject to} & \theta_i := \theta_i - \alpha \frac{\partial j(\theta)}{\partial \theta_i} \\ & \theta_0, \theta_1 \in R, \ i = 1, \cdots, m \end{cases} \tag{4.1}$$

The goal is to reach a point after which cannot move down where it represents the local minimum (optimal point), and one of the important things in the gradient descent algorithm is to choose the learning rate (step length) that determines the speed or slowness in finding the optimal weights values, so the learning rate should be set to suitable values that are not too small because this leads to a gradual descent that will

reach the optimal values, but it will take some time [26], and not large because this may lead to exceeding them, see figure (3.3)

When algorithm of gradient descent is applied continuously to the weights, this eventually leads to reaching optimal weights that reduce the cost function and allow the network to make better predictions [28]. The gradient descent algorithm is presented in the following form;

Algorithm 4.1 Gradient Descent

Require: Learning rate α_n.

Require: The first parameter θ.

While stop criteria not met do

Sample of m examples from the training set $x_1, \cdots, x_m$ with corresponding objectives y_i.

Comput gradient descent estimate: $\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i (h_\theta(x_i) - y_i)$

Applying updates: $\theta \leftarrow \theta - \alpha\hat{g}$

End while

4.1.2 Gradient Descent with Momentum

The gradient descent process can be expanded to include momentum the term (Momentum) refers to a physical quantity in which a negative gradient acts as a force to push the particle through the parameter space in line with Newton's laws of motion, allowing the search to produce inertia in the direction of the search space and suppress noisy oscillations. Momentum demonstrates throwing the ball from the mountain because the speed of the ball increases as it descends [32]. One of the physical quantities is momentum, which according to classical physics is calculated by multiplying an object's mass by its speed. Which forecast not only the direction of objects following a

collision but also their speed because momentum is a vector quantity with a magnitude and direction. Some researchers have proposed the (Momentum) technique, which strengthens oscillations in unrelated directions with improved training in related directions, this method adds a parameter (λ). Momentum only adjusts the relevant sample parameters, reducing the need for useless parameter updates, which leads to faster and more reliable convergence and shrinkage of the oscillation process [12], see figure (4.2)

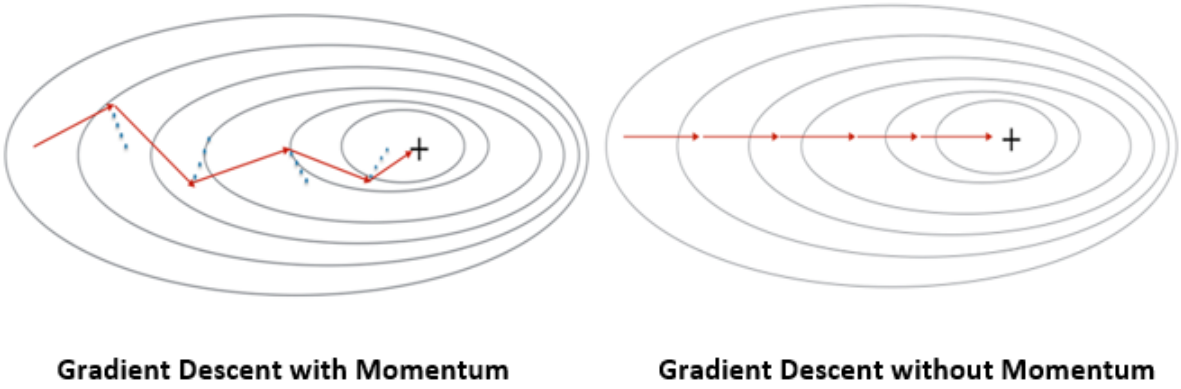


Figure 4.2: G.D and G.D with Momentum
+ It indicates the minimum cost, each step of the descent gradient with momentum is much faster than the gradient in the descent algorithm because it reduces the vertical movement, increases the lateral movement and achieves stability in convergence.

When updating a parameter, Momentum is comparable to adding inertia [17]. The direction and distance of the update are determined by fusing the current gradient with the distance of the direction of the previous update here, another hyperbolic parameter called (Momentum) is employed and represented by the symbol v_t. The update for parameter θ is derived as follows [12];

$$v_t = \lambda v_{t-1} + \alpha \nabla_\theta J(\theta) \qquad \theta := \theta - v_t \tag{4.2}$$

When it use Momentum, we push the ball down the hill Momentum builds up as the ball rolls down and gets faster and faster, despite the air resistance, eventually reaching $\lambda < 1$ because momentum is calculated in time using all previous updates, giving the latest updates more weight in the latest update. When we change our parameters, the Momentum of dimensions whose gradients point in the same directions increases and the update of dimensions whose directions change decreases as a result, which promotes faster convergence and less oscillation [6].

The Momentum algorithm's advantages are the capacity to improve and stabilize network convergence and decrease oscillation, but its drawbacks include the fact that momentum grows along a slope and that the speed at the lowest point may accelerate once again, losing the minimum point [36] [38], the algorithm is presented as follows:

Algorithm 4.2 Gradient Descent with Momentum.

Require: Learning rate α, momentum parameter λ.

Require: The first parameter θ, the first velocity v.

While stop criteria not met do

Sample m examples from training set $x_1, \cdots, x_m$ with corresponding objectives y_i.

Compute gradient descent estimate: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i (h_\theta(x_i), y_i)$

Compute velocity : $v \leftarrow \lambda v + \alpha g$

Applying update: $\theta \leftarrow \theta - v$

End while

Example 4.1.1. Consider objective function;

$$\begin{cases} \text{minimize:} & y = f(x) = 2x^2 \\ \text{subject to} & x_j = x_j - \alpha \dfrac{\partial}{\partial x} f(x) \\ & x \in R,\ j = 1, \cdots, m \end{cases}$$

The partial derivative: $\dfrac{\partial y}{\partial x} = f'(x) = 4x$

By using the update equation for the parameter x, $x_j = x_j - \alpha \dfrac{\partial}{\partial x} f(x)$ Using Python, the above example can be solved, the solution steps are as follows : define the objective function and then calculate its derivative. we apply the gradient descent algorithm first, where it starts with a random point, depending on the search limits (inputs), choose a learning rate ($\alpha = 0.1$) and a number of repetitions (iteration=20), after the example is executed, the drawing of the objective function see figure (4.3), where on the function's curve, the answer shows as a red dot, and a line connects the spots so that we can see how the search descends, see figure (4.4) for the portions of the function with the biggest curvature. The steps are bigger because the gradient (derivative) is bigger. Similar to this, smaller steps are required since the gradient becomes smaller as we approach the ideal solution. The step size is taken into account when scaling the objective function's gradient amount. About 12 iterations were necessary to reach the ideal outcome. Algorithm of gradient descent with Momentum is used the Momentum = 0.3, a learning rate $\alpha = 0.1$, the number of repetitions (iteration= 20) and using the following update equation:

$$v_t = \lambda v_{t-1} + \alpha \nabla_x f(x) \qquad x = x - v_t$$

The optimal solution was found after about 10 iterations, as expected, the application of the gradient descent algorithm with momentum will be faster by finding the optimal solution and fewer iterations, see figure (4.5).The implementation of the above example can be viewed in Python code and display the results with a graphic.

Code 4 (G.D and G.D with Momentum)

```
+ import . . .
def objective(x):
  return 2*x**2.0
r_min, r_max = -1.0, 1.0
inputs = arange(r_min, r_max+0.1, 0.1)
results = objective(inputs)
pyplot.plot(inputs, results)
pyplot.show()
def objective(x):
  return 2*x**2.0
def derivative(x):
  return x * 4.0
def gradient_descent(objective, derivative, bounds, n_iter, step_size):
  solutions, scores = list(), list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
  print('solution=',solution)
  for i in range(n_iter):
   gradient = derivative(solution)
    print('gradient=',gradient )
     solution = solution - step_size * gradient
    print('solution=',solution)
    solution_eval = objective(solution)
    solutions.append(solution)
    scores.append(solution_eval)
    print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
  return [solutions, scores]

seed(4)
bounds = asarray([[-1.0, 1.0]])
n_iter = 20
step_size = 0.1
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_size)
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
results = objective(inputs)
```

```
pyplot.plot(inputs, results)
pyplot.plot(solutions, scores, '.-', color='red’
pyplot.show()
def gradient_descent(objective, derivative, bounds, n_iter, step_size, momentum):
 solutions, scores = list(), list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
 print('solution=',solution )
  change = 0.0
 for i in range(n_iter):
    gradient = derivative(solution)
    print('gradient=',gradient )
    new_change = step_size * gradient + momentum * change
    print('new_change=',new_change )
    solution = solution - new_change
    change = new_change
    solution_eval = objective(solution)
    solutions.append(solution)
    scores.append(solution_eval)
    print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
 return [solutions, scores]
print('solutions=',solutions)
seed(4)
bounds = asarray([[-1.0, 1.0]])
n_iter = 20
step_size = 0.1
momentum = 0.3
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_size,
momentum)
print('solutions=',solutions)
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
results = objective(inputs)
print('results=',results )
pyplot.plot(inputs, results)
pyplot.plot(solutions, scores, '.-', color='orange')
pyplot.show()
```

Out ···

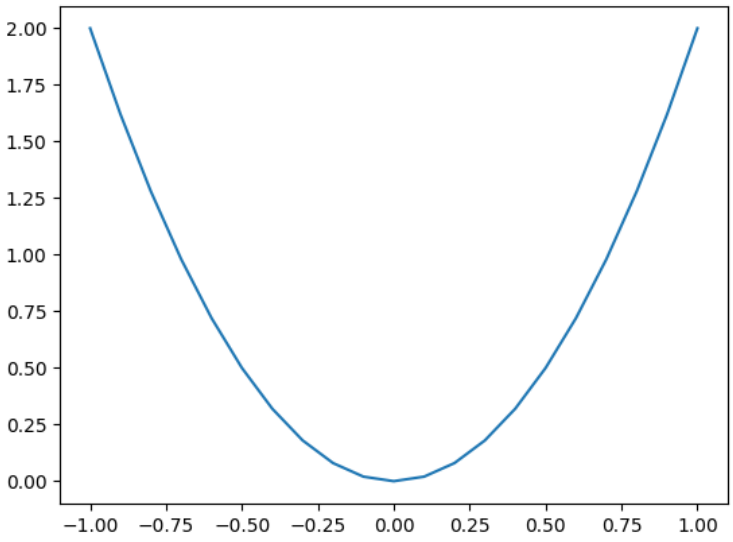


Figure 4.3: $f(x) = 2x^2$ Objective Function

$f(x) = 2x^2$ objective function (optimization function) a simple one-dimensional function squaring the input and determining the range of valid inputs from -1.0 to 1.0 using Python the drawing of the function shows as a parabola.

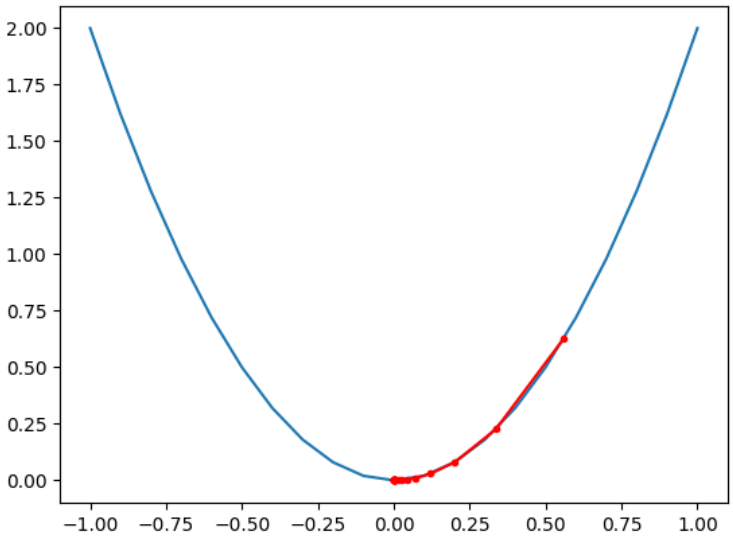


Figure 4.4: Application Gradient Descent

Figure (4.4) it refers to the implementation of the Gradient Descent algorithm on the

objective function (optimization function), which involves starting by choosing an arbitrary point at the boundary of the function, then calculating the Gradient at that point and updating the position in the search space where the solution appears as red dots and connecting the points with a line to see how the search moves down the parts of the function with the larger curve, the Gradient is larger, and therefore more steps are taken, and the Gradient is smaller the closer we get to the optimal solution point (0,0).

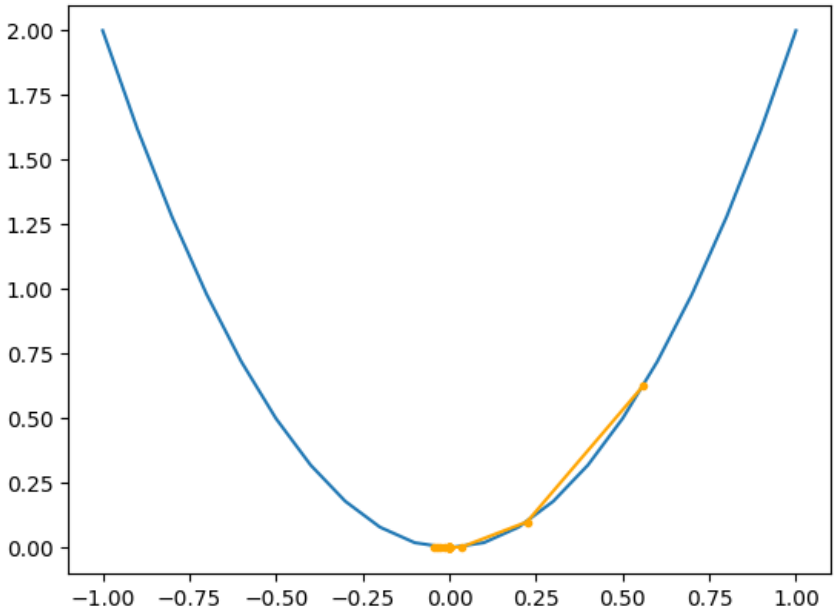


Figure 4.5: Application Gradient Descent with Momentum
Figure (4.5) Indicates the implementation of the Gradient Descent algorithm with Momentum on the objective function (optimization function), the solution appears as orange dots and connect these points with a line to see how the search moves down the algorithm performs a good solution after about 10 iterations as expected is faster and fewer iterations than Gradient Descent without Momentum.

The following table (4.1) shows the value of the random point x, g the value of the G.D, $f(x)$ the value of the function at the random point when applying the G.D algorithm, and v_t the value of the Momentum at the random point, $f_i(x)$ represents the value of the function when applying the G.D algorithm with Momentum.

Table 4.1: Numerical Results of G.D and G.D with Momentum

G.D				**G.D with Momentum**			
i	g	x	$f(x)$	g	v_t	x	$f_i(x)$
0	3.73623	0.56043	0.62818	3.73623	0.37362	0.56043	0.62818
1	2.24174	0.33626	0.22614	2.24174	0.33626	0.22417	0.10051
2	1.34504	0.20175	0.08141	0.89669	0.19054	0.03362	0.00226
3	0.80702	0.12105	0.02931	0.13450	0.07061	−0.03698	0.00274
4	0.48421	0.07263	0.01055	−0.14795	0.00638	−0.04337	0.00376
5	0.29052	0.04357	0.00380	−0.17351	−0.01543	−0.02794	0.00156
6	0.17431	0.02614	0.00137	−0.11177	−0.01580	−0.01213	0.00029
7	0.10459	0.01568	0.00049	−0.04854	−0.00959	−0.00253	0.00001
8	0.06275	0.00941	0.00018	−0.01015	−0.00389	0.00135	0.00000
9	0.03765	0.00564	0.00006	0.00542	−0.00062	0.00198	0.00001
10	0.02259	0.00338	0.00002	0.00792	0.00060	0.00137	0.00000
11	0.01355	0.00203	0.00001	0.00550	0.00073	0.00064	0.00000
12	0.00813	0.00121	0.00000	0.00257	0.00047	0.00016	0.00000
13	0.00487	0.00073	0.00000	0.00066	0.00021	−0.00004	0.00000
14	0.00292	0.00043	0.00000	−0.00017	0.00004	−0.00008	0.00000
15	0.00175	0.00026	0.00000	−0.00035	−0.00002	−0.00006	0.00000
16	0.00105	0.00015	0.00000	−0.00026	−0.00003	−0.00003	0.00000
17	0.00063	0.00009	0.00000	−0.00013	−0.00002	−0.00001	0.00000
18	0.00037	0.00005	0.00000	−0.00004	−0.00001	0.00000	0.00000
19	0.00022	0.00003	0.00000	0.000003	−0.000002	0.000003	0.00000

4.1.3 Nesterov Accelerated Gradient (NAG)

Ilya Suskifer and others broadened the use of Nesterov Momentum in the training of neural networks with gradient descent, emphasizing the significance of initialization and momentum in deep learning, the gradient descent optimization technique has been extended by the concept of Nesterov momentum. The algorithm is known as (Nesterov accelerated gradient) a method for solving the convex programming problem at the convergence rate, a 1983 study by Yuri Nesterov, was the first to provide a brief (NAG) description of the gradient descent optimization process. One of Gradient Descent drawbacks is that if the goal function has noisy gradients, it may become stuck in flat spots or bounce around [36], the optimization process called Gradient Descent (Nesterov Momentum) traces the negative gradient of the function. It is a mechanism that, in terms of speed, smoothes the bouncing gradients and speeds up the search on flat terrain. However, the algorithm may lose the local minimum, continue to grow and exceed the nearest (optimal) solution, if the momentum is high enough [31].

NAG has already demonstrated that it is closely related to classical momentum and only differs in how precisely the velocity vector is updated, although not being considered a type of momentum. In order to perform parameter updating, the method takes a large jump at the beginning based on the prior momentum and then calculates the gradient to correct it. This provides the capacity to predict [17]. Preventing significant oscillations and ensuring that the minimum value solution is found are both benefits of the pre-update method, which is more sensitive to updating parameters.

A greater grasp of the surrounding conditions is gained by moving a little in accordance with the initial solution and then computing the position gradient after the move, allowing for more precise determination of the update's direction, the update for the parameter is as follows;

$$v_t = \lambda v_{(t-1)} + \alpha \nabla_\theta J(\theta - \lambda v_{(t-1)}) \qquad \theta = \theta - v_t \tag{4.3}$$

The term $(\lambda v_{(t-1)})$ refers to the value gradient in the current situation when you update parameter (θ), but the term $(\theta - \lambda v_{(t-1)})$ represents an approximation to the next position of the parameter, after which the future can be predicted by calculating the gradient on the approximate value of the future position of the parameter, to which the term $(\nabla J(\theta - \lambda v_{(t-1)}))$, see figure (4.6)

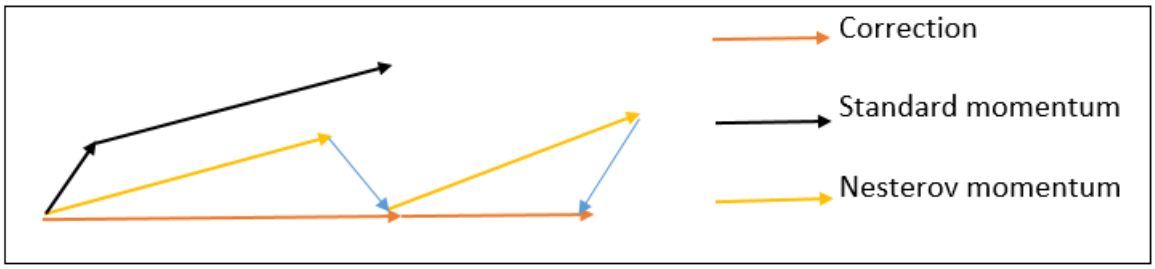


Figure 4.6: The Mechanism of the Accelerated Nesterov Gradient
The update (NAG) makes a big jump in the direction of the previously accumulated gradient (orange line), calculates the gradient, and then corrects it (red line), resulting in a complete update of the black line, which represents momentum, the short line black appears the current gradient, and the long line black, which appears a big jump in the direction of the previous accumulated gradient (NAG). This upgrade prevents us from moving too fast, which enhances responsiveness and performance.

The Nesterov Momentum can be described as a correction factor for the scalar momentum [32] [38].

Algorithm 4.3 Nesterov Accelerated Gradient (NAG)

Require: Parameter of momentum λ, learning rate α.

Require: The first parameter θ, the first velocity v.

While stop criteria not met do

Sample m examples from training set $x_1, \cdots, x_m$ with corresponding objective y_i.

Applying interim update: $\hat{\theta} \leftarrow \theta + \alpha v$

Calculate gradient: $g \leftarrow \frac{1}{m} \nabla_{\hat{\theta}} \sum_i (h(x_i; \hat{\theta}), y_i)$

Calculate velocity update: $v \leftarrow \lambda v + \alpha g$

Applying update: $\theta \leftarrow \theta - v$

End while

4.1.4 Adapative Gradient Descent (Adagrad)

Adagrad algorithm its main goal is to obtain an adaptive rate for each of the weights, which is achieved by adjusting the appropriate learning rate parameters to make smaller updates to the parameters associated with features that occur frequently, but larger updates to the parameters associated with features that occur less frequently [6].
Adagrad algorithm, which determines learning rates for various parameters based on the prior gradients determined for each parameter θ at each time step t, is therefore extremely ideal for processing sparse data. The algorithm steps can be summed up as follows [19];

- Different learning rates should be set for each parameter θ.
- Update parameters corresponding.
- Conducting orientation.

Based on the above, the parameter θ is given the cost function, and the gradient in time step t is denoted by (g_t), and the partial derivative of the objective function (cost) with regard to the parameter θ in time step t is denoted by $g_{t,i}$ where [55],

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

The parameter update form can be written as;

$$G_{t,i} = \sum_{i=0}^{t} (g_{t,i})^2$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}} * g_{t,i} \tag{4.4}$$

Where $G_{t,i}$, it reflects the squared sum of the previous gradients of all parameters θ, ε smoothing factor to prevent division by zero, with a typical value of $\varepsilon = 10^{-3}$.

The advantage of the adagrad algorithm is that it does not require manual adjustment of the learning rate; therefore, if $G_{t,i}$ is smaller in a certain direction, the corresponding learning rate is greater; that is, larger updates are made to the parameters that occur infrequently over time $G_{t,i}$ increases, and the learning rate decreases due to the accumulation of quadratic gradients in the denominator during training, the learning rate declines and becomes infinitely small (each term is added positively). This results in a second vanishing gradient problem, which forces the model to give up entirely on learning [6].

Algorithm 4.4 Adaptive Gradient Descent (Adagrad)

Require: Global rate of learning α.

Require: The first parameter θ.

Require: Small constant $\varepsilon = 10^{-3}$

Adjusting the gradient accumulation parameter G.

While stop criteria not met do

Sample m examples from training set $x_1, \cdots, x_m$ with corresponding objective y_i.

Calculate gradient :$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i (f(x_{i;\theta}), y_i)$

Accumulated squared gradient : $G \leftarrow G + g * g$

Calculate update : $\triangle\, \theta \leftarrow \frac{\alpha}{\sqrt{(G + \varepsilon)}} * g$

Applying update : $\theta \leftarrow + \triangle\, \theta$

End while

4.1.5 Adaptive Delta (AdaDelta)

Adaptive Delta It is an extension of the adagrad ratios optimization technique, published in Matthiw Ziller's 2012 paper (a daptive learning rate method: ADADELTA), the prior adagrad algorithm has a problem with having many iterations and a relatively low learning rate, which causes sluggish convergence [5].

The AdaDelta algorithm has the idea to take a degenerate average to avoid this problem, so it is more powerful, since it limits the range of accumulation and only accumulates the previous gradients to improve efficiency. AdaDelta does not store square gradients, but the average of the previous square gradients, in this way the model continues to learn even after many updates have been made. Hence, the dynamic mean of step t is dependent upon the prior average and the present average, the update form can be written as follows [6];

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

$$E[g^2]_t = \sigma E[g^2]_{t-1} + (1-\sigma) g_t^2$$

$$\triangle\ \theta_t = \frac{-\alpha}{\sqrt{\varepsilon + E[g^2]_t}}\ g_t$$

$$\theta_{t+1} = \theta_t + \triangle\ \theta_t$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{\varepsilon + E[g^2]_t}}\ g_{t,i} \tag{4.5}$$

Where $E[g^2]_{t-1}$ shows mean squared partial derivation of parameter of the previous repetition

$E[g^2]_t$: the decomposed moving mean of the current iteration of the quadratic partial derivative.

g_t^2 : the quadratic partial derivative of the current parameter.

λ : the super parameter usually has a value of 0.9

The algorithm is shown below;

Algorithm 4.5 Adaptive Delta (AdaDelta)

Require: Constant ε, decay rate σ.

Require: The first parameter θ.

Require: Set up the collection of variables $E[g^2]_0 = 0$

While stop criteria not met do

Compute gradient : $g_t = \nabla_\theta J(\theta_t)$

Accumulate gradient squared : $E[g^2]_t = \sigma E[g^2]_{t-1} + (1-\sigma)g_t^2$

Compute $\triangle\ \theta_t = \dfrac{-\alpha}{\sqrt{\varepsilon + E[g^2]_t}}\ g_t$

Applying update : $\theta_{t+1} = \theta_t + \triangle\ \theta_t$

End while

4.1.6 The Root Mean Squared Propagation (RMSProp)

A significant improvement of gradient regression, which is one of the most wide uses methods for fitting deep neural networks, an effective and empirically practical optimization algorithm where it reduces the swing on the vertical axis and moves faster on the horizontal axis, vertical movement leads to a loss of time and effort because the vertical path does not solve any problem or progress towards optimal values [8].

The method is known as square root of the mean squared partial derivatives or Root Mean Square (RMS) [6], because we are using a decaying mean of the partial derivatives and computing the square root of this mean.

Therefore, this algorithm is based on the principle of exponential averages (each value depends on the value before it so that it can converge quickly to reach the optimal values) [36].

This algorithm was invented by Henton which is basically the same as AdaDelta but the

RMSProp algorithm determines the value of $\beta = 0.9$ and the learning rate to the value of $\alpha = 0.01$, RMSProp algorithmits idea on parameter updating consists as follows [10].

$$g_{t,i} = \nabla_\theta j(\theta_{t,i})$$

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}}\, g_t \tag{4.6}$$

As opposed to inefficiently storing square gradients, the total of the gradients is recursively defined as the degenerate average of all prior square gradients, where $E[g^2]_t$ is the average of the decomposed square gradients at time step t. It solely depends on the current gradient and the prior average [10].

$\varepsilon = 10^{-3}$. Is smoothing factor to prevent division by zero (when the value of $E[g^2]_t$ is so small that it is close to zero), $\beta \in [0, 1)$ is hyper-parameter.

This algorithm is based on divide the learning rate of by the current mean of the values of the new gradients of this weight, each parameter in how it is calculated depends on the learning rate and is almost constant, but it calculates the gradient with an exponential average slope instead of the sum of its gradients, so it adapts and meets the variable set learning rates automatically to prevent the learning rate from moving away from the edge and bouncing, which means that the algorithm has excellent performance in unstable problems [6], the algorithm is shown below;

Algorithm 4.6 Root Mean Squared Propagation (RMSProp)

Require: Global rate of learning α , decay- rate λ.

Require: The first parameter θ.

Require: Small constant $\varepsilon = 10^{-3}$

Set up the collection of variables $E[g^2]_0 = 0$

While stop criteria not met do

Sample m examples from training set $x_1, \cdots, x_m$ with corresponding objective y_i.

Calculate gradient : $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i (h(x_i; \theta), y_i)$

Accumulate gradient squared : $E[g^2]_t \leftarrow \lambda E[g^2]_{t-1} + (1 - \lambda) g_t^2$

Calculate parameter update : $\triangle \theta = -\frac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}} * g$

Applying update : $\theta \leftarrow \theta + \triangle \theta$

End while

4.1.7 Adapative Moment Estimation (Adam)

Adam is a first-order optimization algorithm that can be compared to RMSProp and Momentum. Adam calculates the adaptive learning rates for each parameter while also maintaining the average exponential degradation of the former quadratic gradients. Where Momentum is represented by a ball rolling down a slope, Adam represents a heavy ball with friction [10].

The hyperparameters control the exponential mean decay of these moving means, so we calculate the decay averages for the previous squared gradients m_t, v_t as follows [8].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \ \cdots [1]$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \ \cdots [2]$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t} + \varepsilon} \ m_t \tag{4.7}$$

Where α : is the learning rate (step length). $\beta_1, \beta_2 \in [0, 1]$ the exponential rate of decline of the first and second moment estimates. Formula (1) uses parameter β_1 to calculate the mean gradient and formula (2) uses parameter β_2 to calculate the mean squared gradient [11].

m_t : update value of first instant of time in step t.

v_t : update the value of the second instant of time in step t.

m_t, v_t : the coefficients close to zero m_t, v_t are always near zero to solve this problem, the corrected m_t, v_t values are calculated:

$$m_t^{correct} = \frac{m_t}{(1-\beta_1^t)}, \qquad v_t^{correct} = \frac{v_t}{(1-\beta_2^t)} \quad \cdots [3]$$

Then

$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{v_t^{correct} + \varepsilon}} m_t^{correct} \tag{4.8}$$

Algorithm 4.7 Adapative Moment Estimation (Adam)

Require: Learning rate $\alpha = 0.01$

Require: Instant estimations exponential decline rate $\beta_1, \beta_2 \in [0, 1)$ $\beta_1 = 0.9, \beta_2 = 0.99$

Require: Small constant $\varepsilon = 10^{-3}$

Require: The first parameter θ

The first and second instant variables are initialize $v = 0, m = 0$

Time step initialize $t = 0$

While stop not criteria met do

Sample m examples from training set $x_1, \cdots, x_m$ with corresponding objective y_i.

Calculate gradient : $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i (h(x_{i;\theta}, y_i)$

Update $t \leftarrow t + 1$

Update estimate first instant is biased : $m \leftarrow \beta_1 m + (1 - \beta_2) g$

Update estimate second instant is biased : $v \leftarrow \beta_2 v + (1 - \beta_2) g * g$

Correction of the initial bias : $\hat{m} \leftarrow \frac{m}{(1-\beta_1^t)}$

Correction of the initial bias : $\hat{v} \leftarrow \dfrac{v}{(1-\beta_2^t)}$

Calculate update : $\triangle\, \theta = -\dfrac{\hat{m}}{\sqrt{(\hat{v}+\varepsilon)}}$

Applying update : $\theta \leftarrow \theta + \triangle\, \theta$

End while

How to adjust the learning rate?

In the process of training the neural network, reducing the learning rate appropriately will help to improve the speed of learning this type of methods is called low learning rate (gradual decrease) and depends on the number of attempts i.e.(number of repetitions) [36]. If Alpha is selected by a large number, the optimization process will jump wide leaps and will not reach the optimal value, and if we choose an average value for Alpha, the algorithm will not reach it easily and will continue to rotate around itself as it approaches the optimal value, but gradually reducing the alpha value, the algorithm will reach values very close to the optimal value, among the formulas used are as follows [10];

- $\alpha = \dfrac{1}{1 + decay - rate * epoch - number} * \alpha_0$

  Since α_0 is the first value of α,

  decay-rate : a determinable value, let be equal to 1,

  epoch-number : number of attempts.

  Therefore, the greater the number of attempts, the greater the value of the denominator, the lower the final alpha value, and the following table (4.2) shows the relation between the number of attempts and the value of alpha [21].

  $\alpha = \dfrac{1}{(1+1*0)} 0.3 = 0.3$

  $\alpha = \dfrac{1}{(1+1*1)} 0.3 = 0.15$

  $\alpha = \dfrac{1}{(1+1*2)} 0.3 = 0.1 \cdots ect.$

  Steps are initially large, but when we get closer to the ideal value, the alpha value is small and more accurate, these values can be changed by controlling the decay-rate.

- $\alpha = k^{epoch} * \alpha_0 \;\; where \; 0 < k < 1$

  This formula is called the exponential formula, when the number of attempts increases, the value of alpha decrease [21].

- $\alpha = \dfrac{k}{\sqrt{epoch}} * \alpha_0 \;\; where \; 0 < k < 1$

  This formula is called radical formula.

- The peaceful method determines a certain value of alpha for a number of attempts and reduces the value of alpha sequentially.

  $\alpha_0 = 1, \cdots, 100, \;\; \alpha_1 = 100, \cdots, 200, \;\; \alpha_2 = 200, \cdots, 300 \;\; \cdots etc.$

- The manual method in which the Alpha is changed with the monitoring of the processing and this method fixes if the number of data is huge, it takes a number of hours, during these hours and with the monitoring of the processing, a decision is made to increase the alpha or reduce it [5].

4.2 Numerical Results

Example 4.2.1. Suppose that objective function;

$$\begin{cases} \text{minimize} & f(x,y) = x^2 + y^2 \\ \text{subject to} & x, y \in R^n \\ & x, y \geq 0 \end{cases}$$

1-To apply the Nesterov momentum algorithm, the following steps;

- Calculate the partial derivative for each variables.

- Determine input domain for each variable in the objective function where it represents the minimum and upper limit of the variables and determine an arbitrary point in the search boundary.

- Calculate the current position of the point utilizing the gradient descent process with momentum, using the change calculated in the previous iteration. Where;

$$v_{(t+1)} = \lambda v_t - \alpha \nabla f(x, y) \qquad x = x + v_{t+1}$$

- Calculate the expected position of the point $(x - \lambda v_t)$, and calculate its derivative$f'(x - \lambda v_t)$
- Calculate the new location for each variable.

$$v_t = \lambda v_{t-1} - \alpha f'(x - \lambda v_t) \qquad x_t = x_{t-1} + v_t$$

- The algorithm is repeated for each variables in the function.

$iteration = 30,\ step_{size}(\alpha) = 0.01,\ Momentum = 0.3$

These steps can be shown using Python, when applying the example, the target function is drawn in the form of a three-dimensional vessel, then a two-dimensional diagram is made showing the shape of the compact vessel from the inside and showing the solutions in the form of white dots, where each point represents the solution obtained.

Code 5 (Nesterov Accelerated Gradient)

```
+import . . .
def objective(x, y):
   return x ** 2.0 + y ** 2.0
r_min, r_max = -1.0, 1.0
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
figure = pyplot.figure()
axis = figure.add_subplot(projection='3d')
axis.plot_surface(x, y, results, cmap='jet')
pyplot.show()
def objective(x, y):
   return x ** 2.0 + y ** 2.0
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
xaxis = arange(bounds[0, 0], bounds[0, 1], 0.1)
yaxis = arange(bounds[1, 0], bounds[1, 1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
pyplot.show()
def objective(x, y):
   return x ** 2.0 + y ** 2.0
def derivative(x, y):
   return asarray([x * 2.0, y * 2.0])
def nesterov(objective, derivative, bounds, n_iter, step_size, momentum):
   solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
 change = [0.0 for _ in range(bounds.shape[0])]
   for it in range(n_iter):
      projected = [solution[i] + momentum * change[i] for i in range(solution.shape[0])]
      print('projected=',projected)
     gradient = derivative(projected[0], projected[1])
      print('gradient=',gradient)
```

```
 new_solution = list()
      for i in range(solution.shape[0]):
         change[i] = (momentum * change[i]) - step_size * gradient[i]
         print('change[i]=',change[i])
         value = solution[i] + change[i]
         print('value=',value)
          new_solution.append(value)
     solution = asarray(new_solution)
      solution_eval = objective(solution[0], solution[1])
      print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
   return [solution, solution_eval]
seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
step_size = 0.01
momentum = 0.3
best,score= nesterov(objective, derivative, bounds, n_iter, step_size, momentum)
print('Done!')
print('f(%s) = %f' % (best,score))
def nesterov(objective, derivative, bounds, n_iter, step_size, momentum):
  solutions = list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
    change = [0.0 for _ in range(bounds.shape[0])]
    for it in range(n_iter):
      projected = [solution[i] + momentum * change[i] for i in range(solution.shape[0])]
       gradient = derivative(projected[0], projected[1])
      new_solution = list()
      for i in range(solution.shape[0]):
          change[i] = (momentum * change[i]) - step_size * gradient[i]
        print('change[i]=',change[i])
         value = solution[i] + change[i]
         new_solution.append(value)
      solution = asarray(new_solution)
      solutions.append(solution)
```

```
  solution_eval = objective(solution[0], solution[1])
   print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
    return solutions
seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
step_size = 0.01
momentum = 0.3
solutions = nesterov(objective, derivative,bounds, n_iter, step_size, momentum)
xaxis = arange(bounds[0, 0], bounds[0, 1], 0.1)
yaxis = arange(bounds[1, 0], bounds[1, 1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solutions = asarray(solutions)
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
pyplot.show()
```

Out . . .

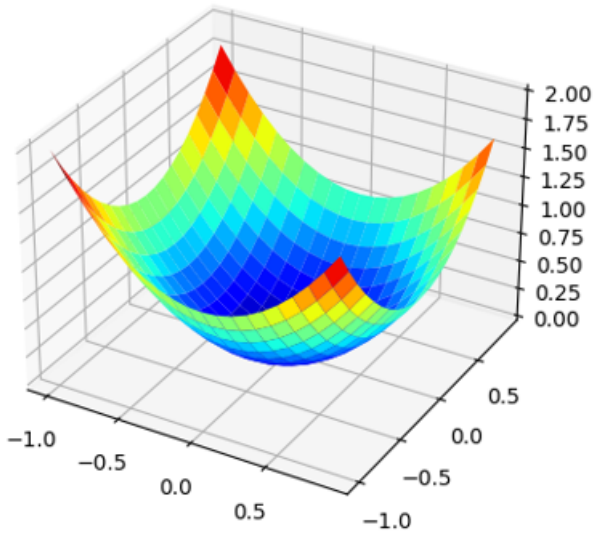


Figure 4.7: Three-dimensional Drawing
The objective function (optimization function) represents a simple two-dimensional function by squaring the inputs and determining the range of inputs between -1.0,1.0 where the three-dimensional figure shows.

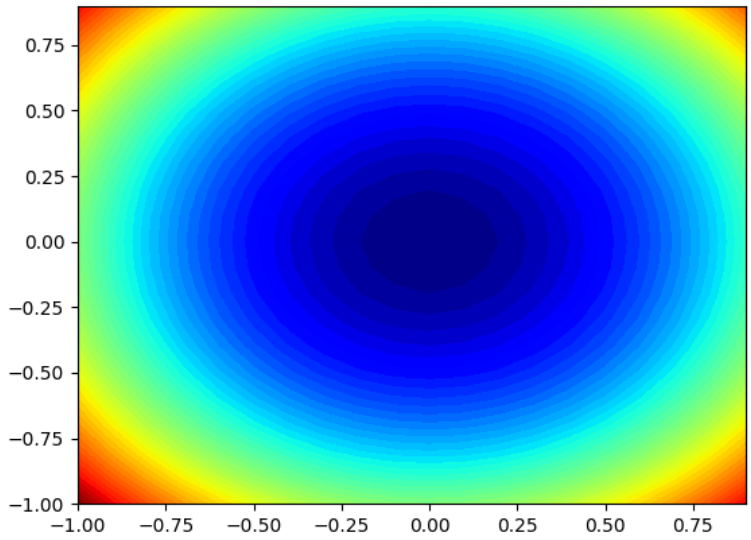


Figure 4.8: Two-dimensional Drawing
It represents a two-dimensional diagram of the function where the shape of the vessel shows a compact of its lines shown in a color gradient (constraints) used to draw points.

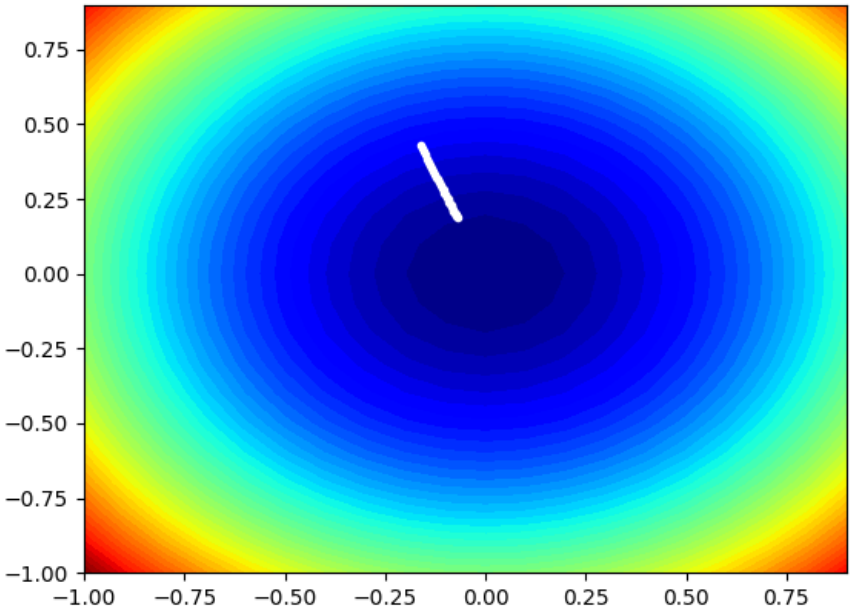


Figure 4.9: Solution points by the NAG Algorithm

The figure (4.9) represents a two-dimensional diagram of the function, where the shape of the vessel shows a compact of its lines shown in a color gradient, the white dots represent the solutions obtained.

Table 4.2: Numerical Results of NAG Algorithm

$p(x)$	$f'(p) = g$	v_{t+1}	x_i	$p(y)$	$f'(p) = g$	v_{t+1}	y_i	$f(x_i, y_i)$
−0.16595	−0.33191	0.00331	−0.16263	0.44064	0.88129	−0.00881	0.43183	0.21293
−0.16164	−0.32328	0.00422	−0.15840	0.42919	0.85838	−0.01122	0.42060	0.20200
−0.15713	−0.31427	0.00441	−0.15399	0.41723	0.83447	−0.01171	0.40889	0.19091
−0.15267	−0.30534	0.00437	−0.14962	0.40538	0.81076	−0.01162	0.39727	0.18021
−0.14830	−0.29661	0.00427	−0.14534	0.39378	0.78757	−0.01136	0.38591	0.17005
−0.14405	−0.28811	0.00416	−0.14117	0.38250	0.76500	−0.01105	0.37485	0.16045
−0.13992	−0.27985	0.00404	−0.13712	0.37153	0.74307	−0.01074	0.36410	0.15138
−0.13591	−0.27182	0.00393	−0.13319	0.36087	0.72175	−0.01044	0.35366	0.14282
−0.13201	−0.26403	0.00382	−0.12937	0.35052	0.70105	−0.01014	0.34351	0.13474
−0.12822	−0.25645	0.00371	−0.12566	0.34047	0.68095	−0.00985	0.33366	0.12712
−0.12455	−0.24910	0.00360	−0.12206	0.33071	0.66142	−0.00956	0.32409	0.11994
−0.12097	−0.24195	0.00350	−0.11855	0.32122	0.64245	−0.00929	0.31480	0.11316
−0.11750	−0.23501	0.00340	−0.11515	0.31201	0.62402	−0.00902	0.30577	0.10676
−0.11413	−0.22827	0.00330	−0.11185	0.30306	0.60612	−0.00876	0.29700	0.10072
−0.11086	−0.22173	0.00320	−0.10864	0.29437	0.58874	−0.00851	0.28848	0.09503
−0.10768	−0.21537	0.00311	−0.10553	0.28592	0.57185	−0.00827	0.28020	0.08965
−0.10459	−0.20919	0.00302	−0.10250	0.27772	0.55545	−0.00803	0.27217	0.08459
−0.10159	−0.20319	0.00293	−0.09956	0.26976	0.53952	−0.00780	0.26436	0.07980
−0.09868	−0.19736	0.00285	−0.09670	0.26202	0.52404	−0.00758	0.25678	0.07529
−0.09585	−0.19170	0.00277	−0.09393	0.25450	0.50901	−0.00736	0.24941	0.07103
−0.09310	−0.18620	0.00269	−0.09124	0.24720	0.49441	−0.00715	0.24226	0.06702

−0.09043	−0.18086	0.00261	−0.08862	0.24011	0.48023	−0.00694	0.23531	0.06323
−0.08783	−0.17567	0.00254	−0.08608	0.23323	0.46646	−0.00674	0.22856	0.05965
−0.08532	−0.17064	0.00246	−0.08361	0.22654	0.453084	−0.00655	0.22201	0.05628
−0.08287	−0.16574	0.00239	−0.08121	0.22004	0.44009	−0.00636	0.21564	0.05310
−0.08049	−0.16099	0.00232	0.07888	0.21373	0.42746	−0.00618	0.20945	0.05010
−0.07818	−0.15637	0.00226	−0.07662	0.20760	0.41520	−0.00600	0.20345	0.04726
−0.07594	−0.15188	0.00219	−0.07442	0.20164	0.40329	−0.00583	0.19761	0.04459
−0.07376	−0.14753	0.00213	−0.07229	0.19586	0.39173	−0.00566	0.19194	0.04207
−0.07165	−0.14330	0.00207	−0.07021	0.19024	0.38049	−0.00550	0.18644	0.03969

The table (4.2) shows the calculated values of each variable x, y and compensating each of them with the objective function using Python.

2-The Adagrad algorithm can be applied to the objective function itself in the following steps;

- Calculate the partial derivative of the function for each variable x, y define the input domain and specify a random point as the starting point.
- Calculation of the gradient for each variable x, y where; $g_{t,i} = \nabla_{x,y} f(x_{t,i}, y_{t,i})$
- To determine the variable's step size, compute sum quadratic partial derivative $G_{t,i} = \sum_i g_{t,i}^2 \qquad \varphi = \dfrac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}}$
  where $\varepsilon = 10^{-3}$ Optional constant to avoid division by zero
- Calculate the update for the variable and then this process is repeated for each variable, $x_{t+1,i} = x_{t,i} - \varphi g_{t,i}$

  The Adagrad algorithm can be implemented using Python, where after execution, a three-dimensional drawing of the function appears, then another drawing representing the shape of the pot compressed from the inside, and then

representing the solution points found on the drawing with white dots, and printing the results;

Code 6 (Adagrad Algorithm)

```
+import . . .
def objective(x, y):
   return x ** 2.0 + y ** 2.0
r_min, r_max = -1.0, 1.0
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
figure = pyplot.figure()
axis = figure.add_subplot(projection='3d')
axis.plot_surface(x, y, results, cmap='jet')
pyplot.show()
def objective(x, y):
   return x ** 2.0 + y ** 2.0
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
xaxis = arange(bounds[0, 0], bounds[0, 1], 0.1)
yaxis = arange(bounds[1, 0], bounds[1, 1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
pyplot.show()
def derivative(x, y):
   return asarray([x * 2.0, y * 2.0])
def adagrad(objective, derivative, bounds, n_iter, step_size):
   solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
   print('solution=',solution)
   sq_grad_sums = [0.0 for _ in range(bounds.shape[0])]
 for it in range(n_iter):
        gradient = derivative(solution[0], solution[1])
      print('gradient=',gradient )
```

```
for i in range(gradient.shape[0]):
        sq_grad_sums[i] += gradient[i] ** 2.0
    print('sq_grad_sums[i]+=',sq_grad_sums[i])
    new_solution = list()
    for i in range(solution.shape[0]):
        alpha = step_size / (1e-3 + sqrt(sq_grad_sums[i]))
        print('alpha=',alpha )
        value = solution[i] - alpha * gradient[i]
        print('value=',value)
        new_solution.append(value)
    solution = asarray(new_solution)
    solution_eval = objective(solution[0], solution[1])
    print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
  return [solution, solution_eval]

seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
step_size = 0.01
best,score = adagrad(objective, derivative, bounds, n_iter, step_size)
print('Done!')
print('f(%s) = %f' % (best,score))
def adagrad(objective, derivative, bounds, n_iter, step_size):
  solutions = list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
  sq_grad_sums = [0.0 for _ in range(bounds.shape[0])]
  for it in range(n_iter):
    gradient = derivative(solution[0], solution[1])
    for i in range(gradient.shape[0]):
      sq_grad_sums[i] += gradient[i]**2.0
    new_solution = list()
    for i in range(solution.shape[0]):
      alpha = step_size / (1e-3 + sqrt(sq_grad_sums[i]))
      value = solution[i] - alpha * gradient[i]
      new_solution.append(value)
```

```
solution = asarray(new_solution)
    solutions.append(solution)
      solution_eval = objective(solution[0], solution[1])
      print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
  return solutions
seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
step_size = 0.01
solutions = adagrad(objective, derivative, bounds, n_iter, step_size)
xaxis = arange(bounds[0,0], bounds[0,1], 0.1)
yaxis = arange(bounds[1,0], bounds[1,1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solutions = asarray(solutions)
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
pyplot.show()
```

Out ···

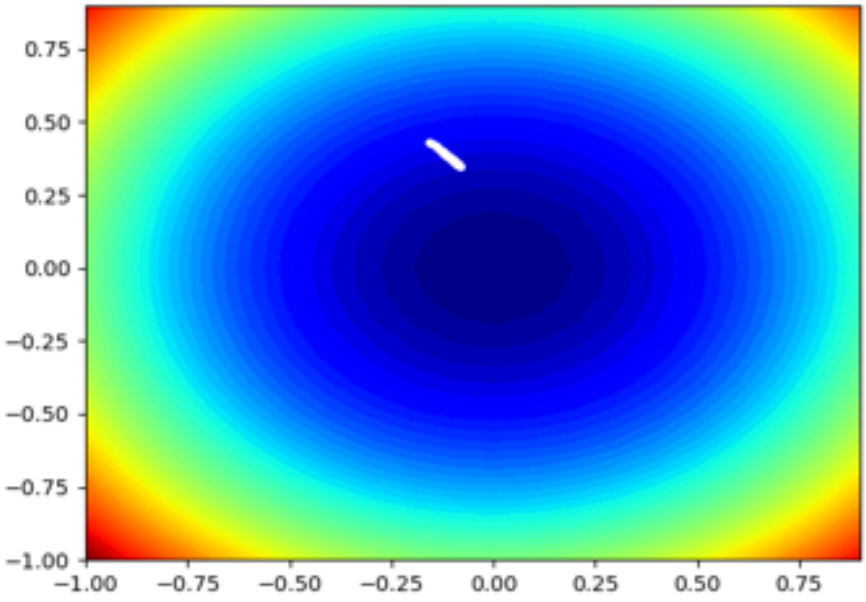


Figure 4.10: Solution points by the Adagrad Algorithm

The figure (4.10) represents a two-dimensional diagram of the function where the shape of the vessel shows a compact of its lines shown in a color gradient the white dots represent the solutions obtained the algorithm needs a large number of iterations to obtain the convergence point of the optimal solution.

Table 4.3: Results of the Adagrad Algorithm

i	$g(x_i)$	$\alpha(x_i)$	x_i	$g(y_i)$	$\alpha(y_i)$	y_i	$f(x_i, y_i)$
0	−0.33191	0.03003	−0.15598	0.88129	0.01133	0.43066	0.20980
1	−0.31197	0.02190	−0.14915	0.86132	0.00810	0.42367	0.20175
2	−0.29830	0.01833	−0.14368	0.84735	0.00668	0.41801	0.19538
3	−0.28736	0.01621	−0.13902	0.83602	0.00583	0.41313	0.19001
4	−0.27804	0.01478	−0.13491	0.82627	0.00525	0.40879	0.18532
5	−0.26982	0.01372	−0.13120	0.81759	0.00482	0.40484	0.18112
6	−0.26241	0.01291	−0.12782	0.80969	0.00449	0.40120	0.17731
7	−0.25564	0.01226	−0.12468	0.80241	0.00422	0.39781	0.17380
8	−0.24937	0.01172	−0.11901	0.79562	0.00382	0.39462	0.17056
9	−0.24352	0.01127	−0.11901	0.78925	0.00382	0.39160	0.16752
10	−0.23803	0.01088	−0.11642	0.78321	0.00366	0.38874	0.16468
11	−0.23284	0.01055	−0.11396	0.77748	0.00352	0.38600	0.16199
12	−0.22793	0.01026	−0.11162	0.77200	0.00339	0.38338	0.15944
13	−0.22325	0.01000	−0.10939	0.76676	0.00328	0.38086	0.15702
14	−0.21878	0.00977	−0.10725	0.76172	0.00318	0.37843	0.15471
15	−0.21451	0.00956	−0.10520	0.75686	0.00310	0.37608	0.15251
16	−0.21041	0.00937	−0.10323	0.75216	0.00301	0.37381	0.14836
17	−0.20646	0.00920	−0.10133	0.74762	0.00294	0.37161	0.14836
18	−0.20266	0.00904	−0.09949	0.74322	0.00287	0.36947	0.14641
19	−0.19899	0.00890	−0.09772	0.73894	0.00281	0.36739	0.14453

20	−0.19545	0.00877	−0.09601	0.73478	0.00275	0.36536	0.14271
21	−0.19202	0.00865	−0.09435	0.73073	0.00270	0.36339	0.14096
22	−0.18870	0.00853	−0.09274	0.72678	0.00265	0.36146	0.13926
23	−0.18548	0.00843	−0.09117	0.72293	0.00260	0.35958	0.13762
24	−0.18235	0.00833	−0.08965	0.71917	0.00255	0.35774	0.13602
25	−0.17931	0.00824	−0.08817	0.71549	0.00251	0.35594	0.13447
26	−0.17635	0.00815	−0.08674	0.71188	0.00247	0.35418	0.13297
27	−0.17348	0.00807	−0.08533	0.70836	0.00244	0.35245	0.13151
28	−0.17067	0.00800	−0.08397	0.70490	0.00240	0.35075	0.13008
29	−0.16794	0.00792	−0.08264	0.70151	0.00237	0.34909	0.12870

$Initial\ value\ of\ x, y = [-0.16595599 \quad 0.44064899] \qquad iteration = 30 \qquad \alpha = 0.01$

The table (4.3) shows the calculated values for each variable x, y and compensating each of them with the objective function using Python, but the values seem far from the convergence point compared to the previous and subsequent algorithms.

3-The AdaDelta algorithm can be applied to the objective function itself in the following steps;

- Calculation of gradient squares for each variable x, y.
  Where $g_{t,i} = \nabla_{x,y} f(x, y),\ sg = [g]_t^2$
- Calculate the average of all square gradients;
  $E[g^2]_t = \beta * E[g^2]_{t-1} + (1 - \beta) * g_t^2$
- Calculation $\Delta x_t = \dfrac{\alpha}{\sqrt{(E[g^2]_t + \varepsilon)}} \qquad \varepsilon = 10^{-3}, \qquad G_{t,ii}$ has been replaced by the decomposed mean on the squared gradients
- Calculation of the new value $x_{t+1} = x_t - \Delta x_t * g_t$
- The algorithm is repeated for each variables in the target function.

The algorithm AdaDelta can be implemented using Python, where after execution a three-dimensional drawing of the function appears, then another drawing representing the shape of the compact vessel from the inside, then representing the solution points on the drawing with white dots, and printing the results;

Code 7 (AdaDelta Algorithm)

```
+import. . .
def objective(x, y):
  return x**2.0 + y**2.0
r_min, r_max = -1.0, 1.0
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
figure = pyplot.figure()
axis = figure.add_subplot(projection='3d')
axis.plot_surface(x, y, results, cmap='jet')
pyplot.show()
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
xaxis = arange(bounds[0,0], bounds[0,1], 0.1)
yaxis = arange(bounds[1,0], bounds[1,1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
pyplot.show()
def derivative(x, y):
  return asarray([x * 2.0, y * 2.0])
def adadelta(objective, derivative, bounds, n_iter, beta, ep=1e-3):
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
  sq_grad_avg = [0.0 for _ in range(bounds.shape[0])]
  sq_para_avg = [0.0 for _ in range(bounds.shape[0])]
  for it in range(n_iter):
    gradient = derivative(solution[0], solution[1])
    for i in range(gradient.shape[0]):
```

```
        sg = gradient[i]**2.0
        print('sg=',sg)
        sq_grad_avg[i] = (sq_grad_avg[i] * beta) + (sg * (1.0-beta))
      print('sq_grad_avg[i]=',sq_grad_avg )
      new_solution = list()
     for i in range(solution.shape[0]):
        alpha = (ep +sqrt(sq_para_avg[i])) / (ep +sqrt(sq_grad_avg[i]))
        print('alpha=',alpha )
         change = alpha * gradient[i]
        print('change=',change)
        sq_para_avg[i] = (sq_para_avg[i] * beta) + (change**2.0 * (1.0-beta))
        print('sq_para_avg[i]=',sq_para_avg )
       value = solution[i] - change
        print('value=',value)
        new_solution.append(value)
       solution = asarray(new_solution)
      solution_eval = objective(solution[0], solution[1])
    print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
   return [solution, solution_eval]
seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
beta = 0.3
best, score = adadelta(objective, derivative, bounds, n_iter, beta)
print('Done!')
print('f(%s) = %f' % (best, score))
def adadelta(objective, derivative, bounds, n_iter, beta, ep=1e-3):
  solutions = list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
   sq_grad_avg = [0.0 for _ in range(bounds.shape[0])]
  sq_para_avg = [0.0 for _ in range(bounds.shape[0])]
   for it in range(n_iter):
    gradient = derivative(solution[0], solution[1])
    for i in range(gradient.shape[0]):
```

```
        sg = gradient[i] ** 2.0
        print('sg=',sg)
         sq_grad_avg[i] = (sq_grad_avg[i] * beta) + (sg * (1.0-beta))
        print('sq_grad_avg[i]=',sq_grad_avg )
      new_solution = list()
      for i in range(solution.shape[0]):
         alpha = (ep + sqrt(sq_para_avg[i])) / (ep +sqrt(sq_grad_avg[i]))
         change = alpha * gradient[i]
        print('change=',change )
         sq_para_avg[i] = (sq_para_avg[i] * beta) + (change ** 2.0 * (1.0 - beta))
        print('sq_para_avg[i]=',sq_para_avg )
         value = solution[i] - change
        print('value=',value)
         new_solution.append(value)
      solution = asarray(new_solution)
      solutions.append(solution)
       solution_eval = objective(solution[0], solution[1])
      print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
    return solutions

seed(1)
solutions = adadelta(objective, derivative, bounds, n_iter, beta)
xaxis = arange(bounds[0, 0], bounds[0, 1], 0.1)
yaxis = arange(bounds[1, 0], bounds[1, 1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solutions = asarray(solutions)
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
pyplot.show()
```

Out $\cdots$

$iteration = 30,$

$step_{size}(\alpha) = 0.01,$

$\beta = 0.3$

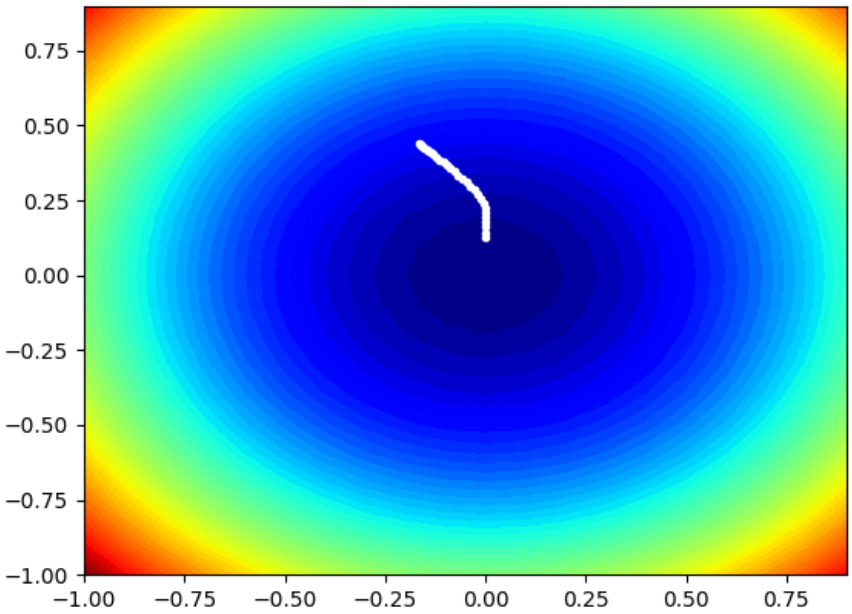


Figure 4.11: Solution points by the AdaDelta Algorithm
The figure(4.11) noted that there was some deviation of the values from the path and then the algorithm began to correct the path to reach the convergence point of the optimal solution.

Table 4.4: Results of the AdaDelta Algorithm

$sg(x)$	α	**change**	x_{t+1}	$sg(y)$	α	**change**	y_{t+1}	$f(x,y)$
0.11016	0.00358	−0.00119	−0.16476	0.77668	0.00135	0.00119	0.43945	0.22027
0.10859	0.00632	−0.00208	−0.16268	0.77248	0.00237	0.00209	0.43736	0.21775
0.10586	0.00874	−0.00284	−0.15983	0.76514	0.00327	0.00286	0.43449	0.21434
0.10219	0.01114	−0.00356	−0.15627	0.75515	0.00414	0.00360	0.43089	0.21010
0.09768	0.01360$	−0.00425	−0.15202	0.74269	0.00501	0.00431	0.42658	0.20508
0.09244	0.01618	−0.00492	−0.14710	0.07278	0.00588	0.00502	0.42155	0.19935
0.08655	0.01891	−0.00556	−0.14153	0.71084	0.00678	0.00571	0.41584	0.19296
0.08013	0.02185	−0.00618	−0.13534	0.69169	0.00769	0.00640	0.40943	0.18596

0.07327	0.02505	−0.00678	−0.12856	0.67055	0.00864	0.00707	0.40236	0.17842
0.06611	0.02857	−0.00734	−0.12122	0.64757	0.00961	0.00774	0.39462	0.17042
0.05877	0.03249	−0.00787	−0.11334	0.62289	0.01063	0.00839	0.38622	0.16202
0.05138	0.03691	−0.00836	−0.10497	0.59668	0.01169	0.00903	0.37719	0.15329
0.04407	0.04196	−0.00881	−0.09616	0.56909	0.01280	0.00966	0.36753	0.14433
0.03699	0.04782	−0.00919	−0.08696	0.54031	0.01397	0.01027	0.35725	0.13519
0.03025	0.05474	−0.00952	−0.07744	0.51052	0.01521	0.01087	0.34638	0.12598
0.02399	0.06305	−0.00976	−0.06767	0.47992	0.01653	0.01145	0.33492	0.11676
0.01832	0.07326	−0.00991	−0.05776	0.44870	0.01794	0.01201	0.32290	0.10761
0.01334	0.08609	−0.00994	−0.04781	0.41707	0.01945	0.01256	0.31034	0.09860
0.00914	0.10269	−0.00982	−0.03799	0.38525	0.02107	0.01308	0.29726	0.08981
0.00577	0.12494	−0.00949	−0.02849	0.35345	0.02284	0.01358	0.28368	0.08129
0.00324	0.15599	−0.00889	−0.01960	0.32190	0.02476	0.01405	0.26962	0.07308
0.00153	0.20146	−0.00789	−0.01171	0.29079	0.02687	0.01449	0.25513	0.06523
0.00054	0.27034	−0.00633	−0.00537	0.26037	0.02920	0.01490	0.24023	0.05774
0.00011	0.37345	−0.00401	−0.00136	0.23084	0.03179	0.01527	0.22495	0.05061
$7.4e-06$	0.49187	−0.00133	$-2.2e-05$	0.20241	0.03469	0.01560	0.20934	0.04383
$1.9e-09$	0.55362	$-2.4e-05$	$2.37e-06$	0.17530	0.03796	0.01589	0.19345	0.03742
$2.2e-011$	0.59940	$2.84e-06$	$-4.7e-07$	0.14969	0.04169	0.01613	0.17731	0.03144
$8.9e-13$	0.66254	$-6.2e-07$	$1.53e-07$	0.12576	0.04598	0.01630	0.16101	0.02593
$9.4e-14$	0.73795	$2.26e-07$	$-7.2e-08$	0.10370	0.05097	0.01641	0.14459	0.02091
$2.1e-14$	0.81388	$-1.1e-07$	$4.58e-08$	0.08363	0.05685	0.01644	0.12815	0.01642

Table (4.4) calculated values for each variable x, y, sg, change and $f(x, y)$ shows the values of the function slightly away from the optimal values, then after updating the parameters, the convergence point for the optimal solution is found in the center of the search.

4-The RMSprop algorithm can be applied to the target function itself the RMSprop algorithm is the same as the AdaDelta algorithm, but the difference between them is the beta value = 0.9, and the solution steps are summarized as follows;

- Calculate the squared gradient; $sg = [g^2]_t$.
- Calculate moving average of the squared gradient;
  $E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)g_t^2$
- Calculate learning rate for this variable; $\lambda = \dfrac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}}$
- Calculate the new value for this variable; $x_{t+1} = x_t - \lambda g_t$
- The algorithm is repeated for each variables in the target function.

Code 8 (RMSProp Algorithm)

```
+import . . .

def objective(x, y):
   return x ** 2.0 + y ** 2.0
r_min,r_max=-1.0,1.0
xaxis=arange(r_min ,r_max ,0.1)
yaxis=arange(r_min ,r_max ,0.1)
x,y=meshgrid(xaxis,yaxis )
results=objective(x,y)
figure=pyplot.figure ()
axis=figure.add_subplot(projection='3d')
axis.plot_surface(x,y,results,cmap='jet')
pyplot .show()
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
xaxis = arange(bounds[0, 0], bounds[0, 1], 0.1)
yaxis = arange(bounds[1, 0], bounds[1, 1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
```

```
pyplot.contourf(x, y, results, levels=50, cmap='jet')
pyplot.show()
def objective(x, y):
  return x**2.0 + y**2.0
def derivative(x, y):
  return asarray([x * 2.0, y * 2.0])
def rmsprop(objective, derivative, bounds, n_iter, step_size, beta):
 solutions = list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
   sq_grad_avg = [0.0 for _ in range(bounds.shape[0])]
  for it in range(n_iter):
     gradient = derivative(solution[0], solution[1])
     for i in range(gradient.shape[0]):
        sg = gradient[i]**2.0
       print('sg=',sg)
        sq_grad_avg[i] = (sq_grad_avg[i] * beta) + (sg * (1.0-beta))
      new_solution = list()
    for i in range(solution.shape[0]):
        lamd  = alpha / sqrt(1e-3 + (sq_grad_avg[i]))
       print('lamd=',lamd )
       value = solution[i] - lamd * gradient[i]
       print('solution[i]=',solution[i])
       print('value=',value)
       new_solution.append(value)
    solution = asarray(new_solution)
     solutions.append(solution)
      solution_eval = objective(solution[0], solution[1])
       print('>%d f(%s) = %.5f' % (it, solution, solution_eval ))
  return solutions
seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
alpha = 0.01
beta = 0.9
```

```
solutions = rmsprop(objective, derivative, bounds, n_iter, alpha, beta)
xaxis = arange(bounds[0,0], bounds[0,1], 0.1)
yaxis = arange(bounds[1,0], bounds[1,1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solutions = asarray(solutions)
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
pyplot.show()
```

Out ···

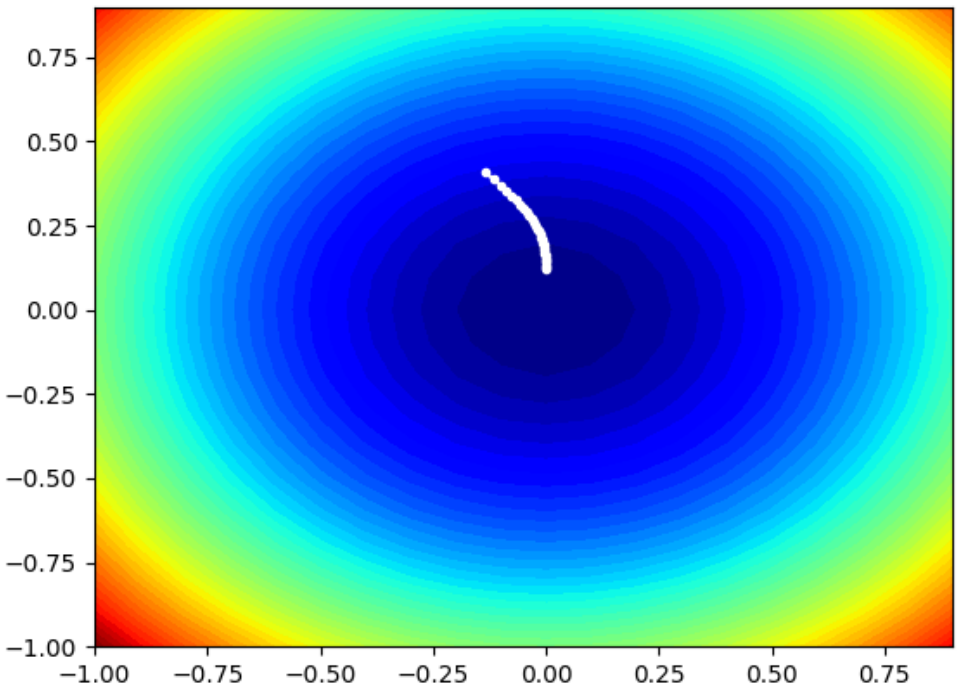


Figure 4.12: Solution points by the RMSProp Algorithm
Figure (4.12) the white dots represent the solutions obtained, and their path seems more stable to reach the convergence point.

Table 4.5: Results of the RMSProp Algorithm

$sg(x)$	$\lambda(x)$	$S(x)$	x_i	$sg(x)$	$\lambda(y)$	$S(y)$	y_i	$f(x_i, y_i)$
0.11016	0.09122	−0.16595	−0.13567	0.77668	0.03565	0.44064	0.40922	0.18588
0.07363	0.07396	−0.13567	−0.11560	0.66986	0.02692	0.40922	0.38718	0.16328
0.05345	0.06757	−0.11560	−0.09998	0.59965	0.02330	0.38718	0.36914	0.14626
0.03998	0.06481	−0.09998	−0.08702	0.54506	0.02130	0.36914	0.35341	0.13247
0.03029	0.06381	−0.08702	−0.07591	0.49960	0.20065	0.35341	0.33923	0.12084
0.02305	0.06388	−0.07591	−0.06621	0.46031	0.01925	0.33923	0.32616	0.11077
0.01753	0.06467	−0.06621	−0.05764	0.42554	0.01871	0.32616	0.31395	0.10189
0.01329	0.06601	−0.05764	−0.05003	0.39427	0.01836	0.31395	0.30242	0.09396
0.01001	0.06780	−0.05003	−0.04325	0.36583	0.01815	0.30242	0.29144	0.08681
0.00743	0.06996	−0.04325	−0.03720	0.33974	0.01804	0.29144	0.28092	0.08030
0.00553	0.07247	−0.03720	−0.03180	0.31567	0.01801	0.28092	0.27080	0.07434
0.00404	0.07529	−0.03180	−0.02701	0.29333	0.01805	0.27080	0.26102	0.06886
0.00291	0.07840	−0.02701	−0.02278	0.27252	0.01815	0.26102	0.25154	0.06379
0.00207	0.08179	−0.02278	−0.01905	0.25309	0.01830	0.25154	0.24233	0.05909
0.00145	0.08544	−0.01905	−0.01579	0.23490	0.01849	0.24233	0.23336	0.05471
0.00099	0.08934	−0.01579	−0.01297	0.21784	0.01873	0.23336	0.22462	0.05062
0.00067	0.09348	−0.01297	−0.01054	0.20182	0.01901	0.22462	0.21608	0.04680
0.00044	0.09785	−0.01054	−0.00848	0.18676	0.01932	0.21608	0.20773	0.04322
0.00028	0.10245	−0.00848	−0.00674	0.17260	0.01967	0.20773	0.19955	0.03987
0.00018	0.10725	−0.00674	−0.00529	0.15929	0.02005	0.19955	0.19155	0.03672
0.00011	0.11226	−0.00529	−0.00410	0.14676	0.00204	0.19155	0.18370	0.03377
$6.75e-05$	0.11746	−0.00410	−0.00314	0.13499	0.02093	0.18370	0.17601	0.03099
$3.95e-05$	0.12284	−0.00314	−0.00237	0.12392	0.02142	0.17601	0.16847	0.02839
$2.24e-05$	0.12838	−0.00237	−0.00176	0.11353	0.02194	0.16847	0.16107	0.02595

$1.24e-05$	0.13409	-0.00176	-0.00128	0.10378	0.02251	0.16107	0.15382	0.02366
$6.65e-06$	0.13995	-0.00128	-0.00092	0.09464	0.02311	0.15382	0.14671	0.02153
$3.45e-06$	0.14593	-0.00092	-0.00065	0.08609	0.02376	0.14671	0.13974	0.01953
$1.73e-06$	0.15203	-0.00065	-0.00045	0.07811	0.02444	0.13974	0.13290	0.01767
$8.37e-07$	0.15824	-0.00045	-0.00031	0.07065	0.02517	0.13290	0.12621	0.01593
$3.91e-07$	0.16452	-0.00031	-0.00020	0.06372	0.02594	0.12721	0.11966	0.01432

Table (4.5) calculated values for each variable $x, y, sg(x), sg(y), \lambda(x), \lambda(y)$ and $f(x, y)$ objective function where the values are close and fairly accurate.

5-The application of Adam's algorithm to the objective function itself can be viewed as a development of both the Adagrad and RMSProp algorithm in the following steps;

- Calculate gradient; $g_t = \nabla_{x,y} f(x, y)$.
- Calculation the estimate of the first moment (average) and second moment (variance) of the gradients;
  $m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t, \qquad v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$
- Calculation; $\hat{v}_t = \dfrac{v_t}{1 - \beta_2^t}, \qquad \hat{m}_t = \dfrac{m_t}{1 - \beta_1^t}$
- Calculate the new value for each variable x, y; $\qquad x_t = x_t - \alpha \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$
- The algorithm is repeated for each variables in the target function.

Code 9 (Adam Algorithm)

```
    +import . . .
    def objective(x, y):
  return x ** 2.0 + y ** 2.0
r_min,r_max=-1.0,1.0
xaxis=arange(r_min ,r_max ,0.1)
yaxis=arange(r_min ,r_max ,0.1)
x,y=meshgrid(xaxis,yaxis )
results=objective(x,y)
figure=pyplot.figure ()
axis=figure.add_subplot(projection='3d')
axis.plot_surface(x,y,results,cmap='jet')
pyplot .show()
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
xaxis = arange(bounds[0, 0], bounds[0, 1], 0.1)
yaxis = arange(bounds[1, 0], bounds[1, 1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
pyplot.show()
def objective(x, y):
  return x ** 2.0 + y**2.0
def derivative(x, y):
  return asarray([x * 2.0, y * 2.0])
```

```
def adam(objective, derivative, bounds, n_iter, alpha, beta1, beta2, eps=1e-3):
    solutions = list()
    x = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
    score = objective(x[0], x[1])
    m = [0.0 for _ in range(bounds.shape[0])]
    v = [0.0 for _ in range(bounds.shape[0])]
    for t in range(n_iter):
      g = derivative(x[0], x[1])
        for i in range(bounds.shape[0]):
          m[i] = beta1 * m[i] + (1.0 - beta1) * g[i]
          print('m[i]=',m[i])
           v[i] = beta2 * v[i] + (1.0 - beta2) * g[i] ** 2
          print('v[i]=',v[i])
          mhat = m[i] / (1.0 - beta1 ** (t + 1))
        vhat = v[i] / (1.0 - beta2 ** (t + 1))
           x[i] = x[i] - alpha * mhat / (sqrt(vhat) + eps)
      score = objective(x[0], x[1])
       solutions.append(x.copy())
      print('>%d f(%s) = %.5f' % (t, x, score))
       print('mhat=',mhat)
       print('vhat=',vhat)
    return solutions
pyplot.show()
seed(1)
```

```
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 30
alpha = 0.01
beta1 = 0.9
beta2 = 0.99
solution = adam(objective, derivative, bounds, n_iter, alpha, beta1, beta2)
xaxis = arange(bounds[0,0], bounds[0,1], 0.1)
yaxis = arange(bounds[1,0], bounds[1,1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solution = asarray(solution)
pyplot.plot(solution[:, 0], solution[:, 1], '.-', color='w')
pyplot.show()
```

Out · · ·

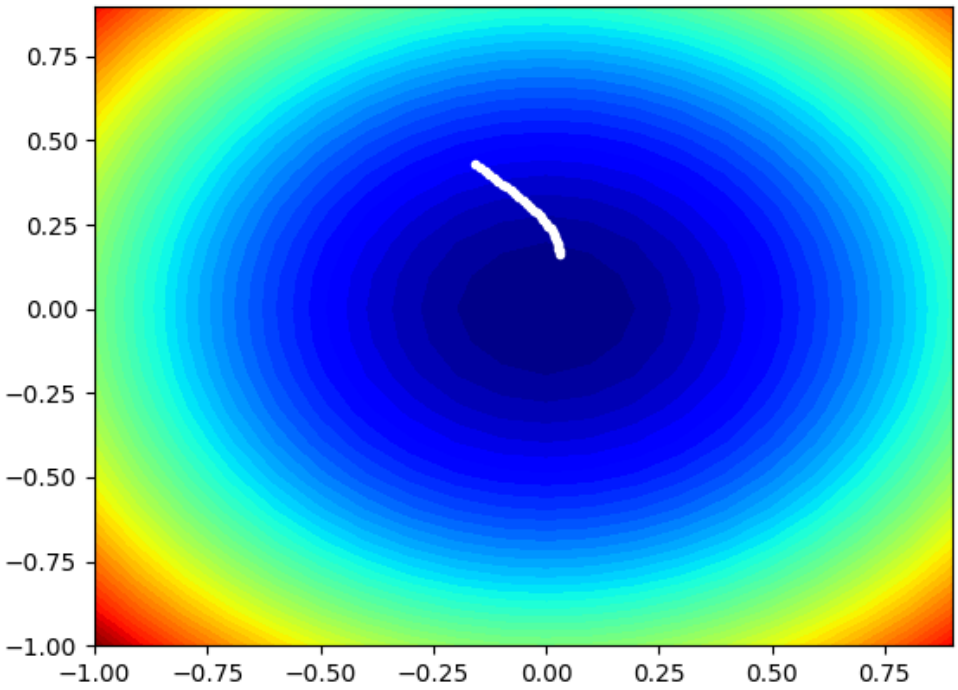


Figure 4.13: Solution points by the Adam Algorithm

The figure (4.13) Adam algorithm the best can be seen that white dots are displayed for each solution found during the search, starting from the top of the optima and gradually approaching the optima in the center of the search and the path of the dots looks more stable and accurate.

Table 4.6: Results of the Adam Algorithm

i	$g(x)$	m_i	v_i	x_t	$\hat{m}_i$	$\hat{v}_i$	x_{t+1}
0	−0.33191	−0.03319	0.00110	−0.16595	−0.33191	0.11016	−0.15598
1	−0.31197	−0.06106	0.00206	−0.15598	−0.32141	0.10371	−0.14603
2	−0.29207	−0.08416	0.00289	−0.14603	−0.31058	0.09751	−0.13612
3	−0.27224	−0.10297	0.00360	−0.13612	−0.29943	0.09157	−0.12625
4	−0.25251	−0.11793	0.00421	−0.12625	−0.28798	0.08590	−0.11646
5	−0.23293	−0.12943	0.00471	−0.11646	−0.27623	0.08049	−0.10676
6	−0.21353	−0.13784	0.00511	−0.10676	−0.26421	0.07535	−0.09717

7	−0.19435	−0.14349	0.00544	−0.09717	−0.25194	0.07049	−0.08772
8	−0.17544	−0.14668	0.00569	−0.08772	−0.23945	0.06590	−0.07843
9	−0.15686	−0.14770	0.00588	−0.07843	−0.22677	0.06158	−0.06932
10	−0.13865	−0.14680	0.00602	−0.06932	−0.21393	0.05753	−0.06044
11	−0.12089	−0.14420	0.00610	−0.06044	−0.20096	0.05375	−0.05181
12	−0.10363	−0.14015	0.00615	−0.05181	−0.18791	0.05024	−0.04346
13	−0.08693	−0.13483	0.00616	−0.04346	−0.17482	0.04699	−0.03544
14	−0.07088	−0.12843	0.00615	−0.03544	−0.16173	0.04399	−0.02776
15	−0.05553	−0.12114	0.00612	−0.02776	−0.14870	0.04124	−0.02048
16	−0.04096	−0.11312	0.00608	−0.02048	−0.13577	0.03872	−0.01361
17	−0.02723	−0.10453	0.00602	−0.01361	−0.12299	0.03642	−0.00720
18	−0.01441	−0.09552	0.00596	−0.00720	−0.11044	0.03434	−0.00127
19	−0.00255	−0.08622	0.00591	−0.00127	−0.09816	0.03245	0.00414
20	0.00828	−0.07677	0.00585	0.00414	−0.08621	0.0307	0.00902
21	0.01805	−0.06729	0.00579	0.00902	−0.07464	0.02922	0.01337
22	0.02674	−0.05789	0.00574	0.01337	−0.06352	0.02783	0.01715
23	0.03430	−0.04867	0.00570	0.01715	−0.05288	0.02659	0.02037
24	0.04075	−0.03972	0.00565	0.02037	−0.04280	0.02547	0.02304
25	0.04608	−0.03114	0.00562	0.02304	−0.03329	0.02445	0.02515
26	0.05031	−0.02300	0.00559	0.02515	−0.02442	0.02353	0.02673
27	0.05347	−0.01535	0.00556	0.02673	−0.01619	0.02269	0.02780
28	0.05561	−0.00825	0.00554	0.02780	−0.00866	0.02191	0.02838
29	0.05677	−0.00175	0.00551	0.02838	−0.00182	0.02119	0.02851

The table (4.6) represents the values of the variable x, the values of m_i, v_i and the updated values of m_i, v_i, x_{t+1}.

Table 4.7: Results of the Adam Algorithm

i	$g(y)$	m_i	v_i	y_t	$\hat{m}_i$	$\hat{v}_i$	y_{t+1}	$f(x,y)$
0	0.88129	0.08812	0.00776	0.44064	0.88129	0.77668	0.43066	0.20980
1	0.86132	0.16544	0.1510	0.43066	0.87078	0.75919	0.42067	0.19830
2	0.84135	0.23303	0.02203	0.42067	0.85992	0.74191	0.41070	0.18721
3	0.82141	0.29187	0.02856	0.41070	0.84872	0.72486	0.40074	0.17654
4	0.80149	0.34283	0.03470	0.40074	0.83719	0.70803	0.39081	0.16630
5	0.78162	0.38671	0.04046	0.39081	0.82533	0.69144	0.38089	0.15648
6	0.76179	0.42422	0.04586	0.38089	0.81315	0.67508	0.37101	0.14709
7	0.74202	0.45600	0.05090	0.37101	0.80066	0.65897	0.36116	0.13813
8	0.72232	0.48263	0.05561	0.36116	0.78787	0.64310	0.35134	0.12960
9	0.70269	0.50464	0.05999	0.35134	0.77479	0.62749	0.34158	0.12148
10	0.68316	0.52249	0.06406	0.34158	0.76144	0.61212	0.33186	0.11379
11	0.66372	0.53661	0.06783	0.33186	0.74782	0.59702	0.32219	0.10649
12	0.64439	0.54739	0.07130	0.32219	0.73395	0.58218	0.31258	0.09960
13	0.62517	0.55517	0.07450	0.31258	0.71985	0.56760	0.30304	0.09309
14	0.60609	0.56026	0.07742	0.30304	0.70552	055329	0.29357	0.08696
15	0.58714	0.56295	0.08010	0.29357	0.69099	0.53925	0.28417	0.08118
16	0.56835	0.56349	0.08253	0.28417	0.67627	0.52548	0.27486	0.07573
17	0.54972	0.56211	0.08472	0.27486	0.66138	0.51199	0.26563	0.07061
18	0.53126	0.55903	0.08670	0.26563	0.64634	0.49877	0.25649	0.06579
19	0.51298	0.55442	0.08846	0.25649	0.63116	0.48583	0.24744	0.06125
20	0.49489	0.54847	0.09003	0.24744	0.61586	0.47317	0.23850	0.05697
21	0.47701	0.54132	0.09140	0.23850	0.60045	0.46079	0.22967	0.05293
22	0.45935	0.53313	0.09260	0.22967	0.58497	0.44869	0.22095	0.04912
23	0.44191	0.52400	0.09363	0.22095	0.56943	0.43686	0.21235	0.04551

24	0.42470	0.51407	0.09449	0.21235	0.55383	0.42532	0.20387	0.04210
25	0.40775	0.50344	0.09521	0.20387	0.53822	0.41405	0.19552	0.03886
26	0.39104	0.49220	0.09579	0.19552	0.52259	0.40306	0.18730	0.03580
27	0.37461	0.48044	0.09623	0.18730	0.50697	0.39235	0.17922	0.03289
28	0.35844	0.46824	0.09656	0.17922	0.49139	0.38192	0.17128	0.03014
29	0.34257	0.45567	0.09676	0.17128	0.47585	0.37175	0.16349	0.02754

The table (4.7) represents the values of the variable y, the values of m_i, v_i and the updated values of m_i, v_i, y_{t+1}.

6-The following table shows the values of the objective function after applying both algorithm NAG, Adagrad, AdaDelta, RMSProp and Adam. Where $iteration = 30$, $step - size(\alpha) = 0.01$, $Momentum = 0.3$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\varepsilon = 10^{-3}$.

Table 4.8: Results of comparing the values of the objective function by applying algorithms

i	**NAG f(x,y)**	**Adagrad f(x,y)**	**AdaDelta f(x,y)**	**RMSProp f(x,y)**	**Adam f(x,y)**
0	0.21293	0.20980	0.22027	0.18588	0.20980
1	0.20200	0.20175	0.21775	0.16328	0.19830
2	0.19091	0.19538	0.21434	0.14626	0.18721
3	0.18021	0.19001	0.21010	0.13247	0.17654
4	0.17005	0.18532	0.20508	0.12084	0.16630
5	0.16045	0.18112	0.19935	0.11077	0.15648
6	0.15138	0.17731	0.19296	0.10189	0.14709
7	0.14282	0.17380	0.18596	0.09396	0.13813
8	0.13474	0.17056	0.17842	0.08681	0.12960

i	NAG f(x,y)	Adagrad f(x,y)	AdaDelta f(x,y)	RMSProp f(x,y)	Adam f(x,y)
9	0.12712	0.16752	0.17042	0.08030	0.12148
10	0.11994	0.16468	0.16202	0.07434	0.11379
11	0.11316	0.16199	0.15329	0.06886	0.10649
12	0.10676	0.15944	0.14433	0.06379	0.09960
13	0.10072	0.15702	0.13519	0.05909	0.09309
14	0.09503	0.15471	0.12598	0.05471	0.08696
15	0.08965	0.15251	0.11676	0.05062	0.08118
16	0.08459	0.15039	0.10761	0.04680	0.07573
17	0.07980	0.14836	0.09860	0.04322	0.07061
18	0.07529	0.14641	0.08981	0.03987	0.06579
19	0.07103	0.14453	0.08129	0.03672	0.06125
17	0.07980	0.14836	0.09860	0.04322	0.07061
18	0.07529	0.14641	0.08981	0.03987	0.06579
19	0.07103	0.14453	0.08129	0.03672	0.06125
20	0.06702	0.14271	0.07308	0.03377	0.05697
21	0.06323	0.14096	0.06523	0.03099	0.05293
22	0.05965	0.13926	0.05774	0.02839	0.04912
23	0.05628	0.13762	0.05061	0.02595	0.04551
24	0.05310	0.13602	0.04383	0.02366	0.04210
25	0.05010	0.13447	0.03742	0.02153	0.03886
26	0.04726	0.13297	0.03144	0.01953	0.03580
27	0.04459	0.13151	0.02593	0.01767	0.03289
28	0.04207	0.13008	0.02091	0.01593	0.03014
29	0.03969	0.12870	0.01642	0.01432	0.02754

Through the table (4.8), it was shown that the value of the function using the NAG algorithm points began to descend slowly, and then began to descend rapidly to reach the optimal value due to the accumulation of momentum, it may exceed it and skip it when the number of repetitions increases.

While adagrad was some what slow at the beginning of training because the update is large, but in the late period of training, the update was small due to the accumulation of the gradient box, which led to a decrease in the learning rate prematurely.

Adadelta, RMSProp, Adam its performance are better to reach the optimal value it is clear that the descent is gradual in the values of the objective function, but the AdaDelta algorithm began to descend to the lower position gradually and in slow jumps at the beginning of training and then began to accelerate little by little or, perhaps, slightly move away from the search path. RMSProp had big jumps and faster to reach the acceptable solution, but its accuracy is low because it may skip over some important values and exceed them.

Adam's algorithm relied on the centers of power in adagrad, Adadelta and RMSProp, and the points began to descend gradually and with convergent and comprehensive values, so it is effective and computationally efficient and requires little adjustment for super parameters, the following diagram shows the comparison of Adam with other optimization algorithms.

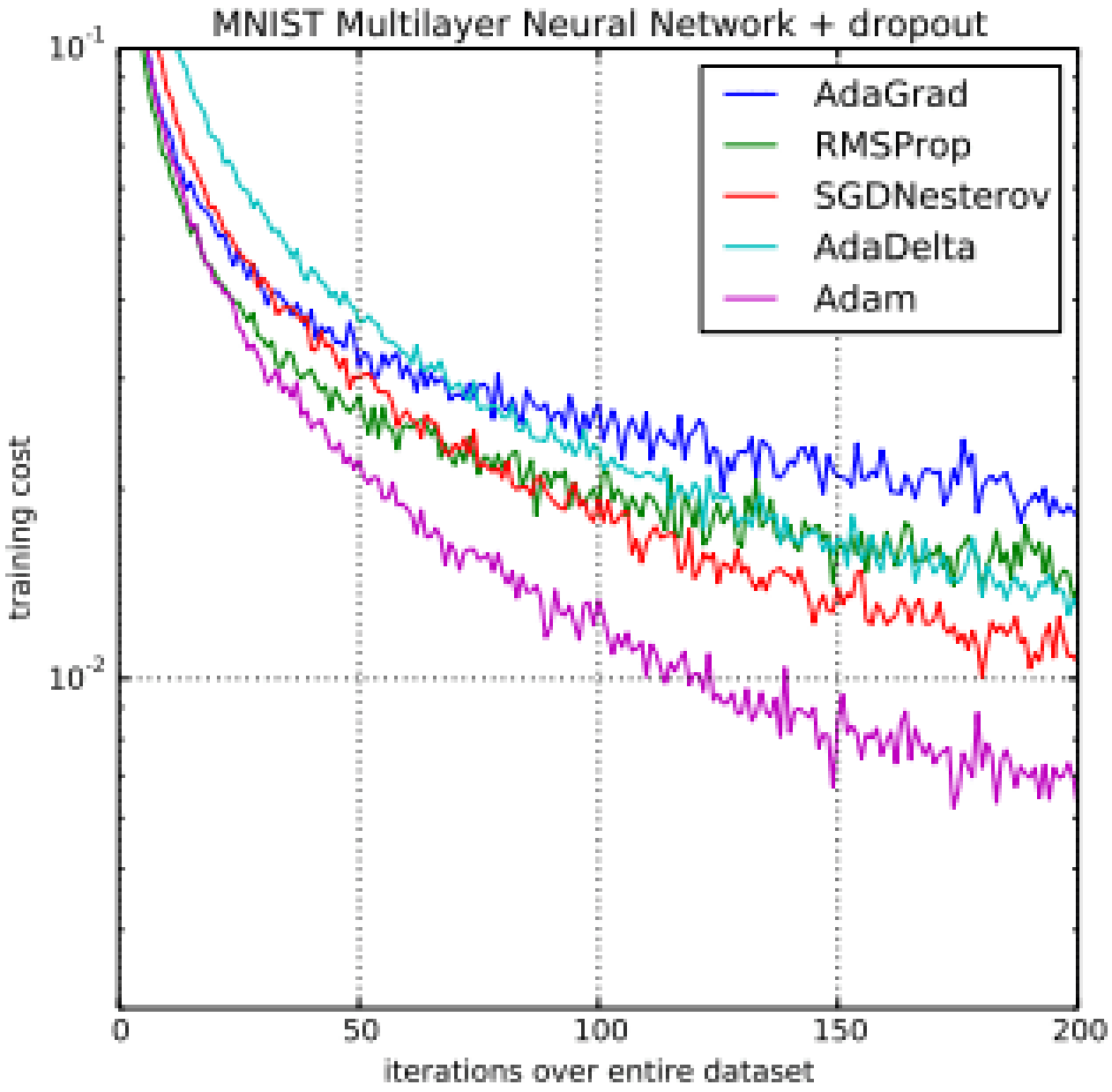


Figure 4.14: Comparison of Adam and other Algorithms [10]

4.3 Conclusions:.

1-The Gradient Descent algorithm is good for simple data, it is easy to calculate, implement, and understand, but if the data is large, it may take a long time to converge from the minimum and reach the optimal value.

2-The Gradient Descent algorithm with Momentum is good in that it reduces the vertical (longitudinal) movement, which will limit the oscillations to some extent and speeds up the horizontal (lateral) movement, so you can get faster convergence and less oscillations, but you may lose the optimal value due to the accumulation of momentum (speed).

3-Nesterov's methodology calculates the gradient after applying the current velocity, therefore it may be thought of as a correction factor added to the conventional momentum method.

4-The RMSProp algorithm is a useful and efficient optimization technique for neural networks that deep learning experts frequently utilize.

5-This Adagrad approach workseffectively in some deep learning models, but not all, since it was experimentally discovered that the accumulation of the gradient square from the beginning of training will lead to an early and excessive decrease in the effective learning rate.

6-RMSProp, Adadelta , and similar Adam are good algorithms, Adam performs better when inspecting the offset.

7-The best optimization algorithm is Adam, which performs admirably for all deep learning problems, particularly after setting the super parameters because it is built on the advantages of the prior algorithms, if not the best ones, and because it provides better performance in terms of cost and quick and high performance.

8-Adam reviews the initial estimates of the first moment (average) and the second moment (offset). Because the RMSProp also uses the second moment estimation, but the correction coefficient is missing. Adam is using correction coefficients.

9-Adam's technique can cope with sparse gradients in noise problems, is easy to use, efficient and memory-light. It is also excellent for non-static targets (e.g. RMSProp) and sparse gradients problems (e.g. Adagrad) it is the fastest convergence algorithm with minimum.

4.4 Future Work

1- Artificial Neural Networks (ANNs) algorithms are great and good for some tasks and not good for others.

2- Gradient Descent algorithm is sufficient for simple tasks.

3- Adagrad algorithm performs well when applied to convex problems and problems involving sparse gradients.

4- RMSProp it is suitable for non-static targets.

5- Adam the best algorithm optimized for neural networks. Adam performs very well for any deep learning problem that we may encounter as it can handle sparse gradients in noise problems.

6-Super parameters can be set to other values, get results, compare them, determine the best ones, increase the number of repetitions to 100 or more to reach the optimal value of the objective function when $f(x, y) = 0$ and set the parameters to

$\beta_1 = 0.99 \qquad \beta_2 = 0.999 \qquad step - size = 0.001$

7-The mean squared error can be calculated for each algorithm and the statement that the least error ratio is the closest to the optimal values.

and these algorithms are still in continuous development $\cdots$

REFERENCES

[1] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[2] Niclas Andréasson, Anton Evgrafov, and Michael Patriksson. *An introduction to continuous optimization: foundations and fundamental algorithms*. Courier Dover Publications, 2020.

[3] Vladimir I Arnold. *Ordinary differential equations*. Springer Science & Business Media, 1992.

[4] Rajesh Kumar Arora. *Optimization: algorithms and applications*. CRC press, 2015.

[5] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[6] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[7] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

[8] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.

[9] William E Boyce, Richard C DiPrima, and Douglas B Meade. *Elementary differential equations and boundary value problems*. John Wiley & Sons, 2021.

[10] Jason Brownlee. *Optimization for machine learning*. Machine Learning Mastery, 2021.

[11] Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*, volume 75. John Wiley & Sons, 2013.

[12] Richard E Cutkosky. Singularities and discontinuities of feynman amplitudes. *Journal of Mathematical Physics*, 1(5):429–433, 1960.

[13] George Bernard Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1998.

[14] Jay Dawani. *Hands-On Mathematics for Deep Learning: Build a solid mathematical foundation for training efficient deep neural networks*. Packt Publishing Ltd, 2020.

[15] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[16] Robert Dorfman, Paul Anthony Samuelson, and Robert M Solow. *Linear programming and economic analysis*. Courier Corporation, 1987.

[17] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[18] Leslie R Foulds. *Optimization techniques: an introduction*. Springer Science & Business Media, 2012.

[19] James A Freeman and David M Skapura. *Neural networks: algorithms, applications, and programming techniques*. Addison Wesley Longman Publishing Co., Inc., 1991.

[20] Saul I Gass. *Linear programming: methods and applications*. Courier Corporation, 2003.

[21] Shuzhi Sam Ge, Chang C Hang, Tong H Lee, and Tao Zhang. *Stable adaptive neural network control*, volume 13. Springer Science & Business Media, 2013.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[23] B Guenin, J Könemann, and L Tunçel. *A Gentle Introduction to Optimization*. Cambridge University Press, 2014.

[24] Magnus Lie Hetland. *Python Algorithms: mastering basic algorithms in the Python Language*. Apress, 2014.

[25] Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *Siam review*, 61(4):860–891, 2019.

[26] Edward L Ince. *Ordinary differential equations*. Courier Corporation, 1956.

[27] David S Johnson. A catalog of complexity classes. In *Algorithms and complexity*, pages 67–161. Elsevier, 1990.

[28] John D Kelleher. *Deep learning*. MIT press, 2019.

[29] Cornelius Lanczos. Solution of systems of linear equations by minimized iterations1. *Journal of Research of the National Bureau of Standards*, 49(1), 1952.

[30] Fetty Fitriyanti Lubis, Yusep Rosmansyah, and Suhono Harso Supangkat. Gradient descent and normal equations on cost function minimization for online predictive using linear regression with multiple variables. In *2014 International Conference on ICT For Smart Society (ICISS)*, pages 202–205. IEEE, 2014.

[31] Sean Luke. *Essentials of metaheuristics*, volume 2. Lulu Raleigh, 2013.

[32] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade*, volume 7700. springer, 2012.

[33] Berndt Müller, Joachim Reinhardt, and Michael T Strickland. *Neural networks: an introduction*. Springer Science & Business Media, 1995.

[34] Yasuo Murata. *Mathematics for stability and optimization of economic systems*. Academic Press, 2014.

[35] Phil Picton and Phil Picton. *What is a neural network?* Springer, 1994.

[36] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[37] R Clark Robinson. Introduction to mathematical optimization. *Department of Mathematics, Northwestern University, Illinois US*, 2013.

[38] Albert Schrotenboer. *Exact and heuristic methods for optimization in distributed logistics*. PhD thesis, University of Groningen, SOM research school, 2020.

[39] Subana Shanmuganathan. *Artificial neural network modelling: An introduction*. Springer, 2016.

[40] Simone Sharma and Anidhya Athaiya. Activation functions in neural networks.

[41] Jan A Snyman and Daniel N Wilke. *Practical Mathematical Optimization: Basic Optimization Theory and Gradient-Based Algorithms*, volume 133. Springer, 2018.

[42] Michael Stolz. The history of applied mathematics and the history of society. *Synthese*, 133:43–57, 2002.

[43] Jalaj Thanaki. *Python natural language processing*. Packt Publishing Ltd, 2017.

[44] Radha Thangaraj, Millie Pant, Ajith Abraham, and Pascal Bouvry. Particle swarm optimization: Hybridization perspectives and experimental illustrations. *Applied Mathematics and Computation*, 217(12):5208–5226, 2011.

[45] Ioannis G Tsoulos and Athanassios Stavrakoudis. Enhancing pso methods for global optimization. *Applied Mathematics and Computation*, 216(10):2988–3001, 2010.

[46] Jean van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, volume 9. Harvard University Press, 1967.

[47] Guido VanRossum and Fred L Drake. *The python language reference*. Python Software Foundation Amsterdam, Netherlands, 2010.

[48] JIŘÍ VOHRADSKY. Neural network model of gene expression. *the FASEB journal*, 15(3):846–854, 2001.

[49] Philip D Wasserman and Tom Schwartz. Neural networks. ii. what are they and why is everybody so interested in them now? *IEEE expert*, 3(1):10–15, 1988.

[50] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[51] Xin-She Yang. *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons, 2010.

[52] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[53] Xin-She Yang. Optimization algorithms. *Computational optimization, methods and algorithms*, pages 13–31, 2011.

[54] Xin-She Yang. Optimization and metaheuristic algorithms in engineering. *Metaheuristics in water, geotechnical and transport engineering*, 1:23, 2013.

[55] Xin-She Yang. *Introduction to algorithms for data mining and machine learning*. Academic press, 2019.

[56] Xiang-Sun Zhang. *Neural networks in optimization*, volume 46. Springer Science & Business Media, 2013.

[57] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

[58] A Žilinskas. Practical mathematical optimization: An introduction to basic optimization theory and classical and new gradient-based algorithms, 2006.

الملخص

الشبكات العصبية الاصطناعية هي منهجية حديثة ومتطورة جذبت اهتمام العديد من الباحثين في مختلف المجالات، بما في ذلك الهندسة وتكنولوجيا المعلومات والطب والإحصاء وبحوث العمليات، وتكتسب أهمية متزايدة نظرا لمرونتها الكبيرة مقارنة بالطرق التقليدية، فضلا عن قدرتها على التعلم والتكيف مع أي نموذج.

عادة ما يتم تدريب الشبكات العصبية باستخدام أساليب وطرق قائمة على التدرج والتكرار لمحاولة تقليل وظيفة التكلفة، حيث يقوم النموذج بضبط وزنه ومنحدره، بالاعتماد على وظيفة التكلفة والتعلم المعزز (التعلم الخاضع للإشراف) للوصول إلى نقطة التقارب (القيمة المثلى).

أخذت هذه الدراسة جانبين، الأول هو دراسة الشبكات العصبية الخطية، بأبسط أنواعها تستخدم في دراسة (الانحدار الخطي، الانحدار الخطي المتعدد، الانحدار متعدد الحدود، الانحدار اللوجستي) باستخدام طريقة النسب المتدرج التكرارية والمعادلة العادية كنهج تحليلي للتحسين أو كبديل لنزول التدرج من أجل تقليل دالة التكلفة، والوصول إلى نقطة التقارب للحد الأدنى.

أما الجانب الآخر من الدراسة فهو كيفية تحسين وتطوير خوارزميات الشبكات العصبية من خلال إجراء مقارنات مع بعضها البعض واختيار أفضلها من حيث السرعة في الأداء والتكاليف المنخفضة والأكثر دقة وأفضل النتائج كما قمنا بتحليلها وأظهار نقاط القوة والضعف لكل منها. لقد ثبت أن آدم يقوم بتدريب النموذج بشكل جيد لأنه يعتمد على مراكز القوة في الخوارزميات السابقة، فقد تم حساب نتائجه باستخدام بايثون من خلال تطبيق أربعة برامج في وقت واحد، حيث يظهر البرنامج الأول رسما ثلاثي الأبعاد للدالة الموضوعية، ويظهر البرنامج الثاني شكل الدالة المضغوطة من الأعلى ويشير إلى تدرجات اللون كقيود للدالة، ثم يطبع النتائج ويعرض الحل كنقاط بيضاء في مسار البحث، حيث تمثل كل نقطة حلا يتم الحصول عليه من خلال تطبيق الخوارزمية.

جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة بابل
كلية التربية للعلوم الصرفة
قسم الرياضيات

تطوير خوارزميات التحسين الرياضي في الشبكات العصبية

رسالة

مقدمة الى مجلس كلية التربية للعلوم الصرفة / جامعة بابل وهي جزء من متطلبات نيل درجة الماجستير في التربية / الرياضيات

من قبل الطالبة
وئام عباس عبيد مطر
باشراف
أ.م.د. احمد صباح احمد الجيلاوي

1444 هـ 2023 م